DRM Specification

Approved Version 2.1 – 06 Nov 2008

Open Mobile Alliance

OMA-TS-DRM-DRM-V2_1-20081106-A

Each Open Mobile Alliance member has agreed to use reasonable endeavors to inform the Open Mobile Alliance in a timely manner of Essential IPR as it becomes aware that the Essential IPR is related to the prepared or published specification. However, the members do not have an obligation to conduct IPR searches. The declared Essential IPR is publicly available to members and non-members of the Open Mobile Alliance and may be found on the "OMA IPR Declarations" list at http://www.openmobilealliance.org/ipr.html. The Open Mobile Alliance has not conducted an independent IPR review of this document and the information contained herein, and makes no representations or warranties regarding third party IPR, including without limitation patents, copyrights or trade secret rights. This document may contain inventions for which you must obtain licenses from third parties before making, using or selling the inventions. Defined terms above are set forth in the schedule to the Open Mobile Alliance Application Form.

NO REPRESENTATIONS OR WARRANTIES (WHETHER EXPRESS OR IMPLIED) ARE MADE BY THE OPEN MOBILE ALLIANCE OR ANY OPEN MOBILE ALLIANCE MEMBER OR ITS AFFILIATES REGARDING ANY OF THE IPR'S REPRESENTED ON THE "OMA IPR DECLARATIONS" LIST, INCLUDING, BUT NOT LIMITED TO THE ACCURACY, COMPLETENESS, VALIDITY OR RELEVANCE OF THE INFORMATION OR WHETHER OR NOT SUCH RIGHTS ARE ESSENTIAL OR NON-ESSENTIAL.

THE OPEN MOBILE ALLIANCE IS NOT LIABLE FOR AND HEREBY DISCLAIMS ANY DIRECT, INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR EXEMPLARY DAMAGES ARISING OUT OF OR IN CONNECTION WITH THE USE OF DOCUMENTS AND THE INFORMATION CONTAINED IN THE DOCUMENTS.

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) is to enable the distribution and consumption of digital content in a controlled manner. The content is distributed and consumed on authenticated Devices per the usage rights expressed by the content owners. OMA DRM work addresses the various technical aspects of this system by providing appropriate specifications for content formats, protocols, and a rights expression language.

A number of DRM specifications have already been defined within the OMA. See [DRM], [DRMDCF] and [DRMREL]. These existing specifications are referred to within this document as “release 1”.

This specification defines the mechanisms and protocols necessary to implement the OMA DRM release 2.1 system. The specification addresses specific requirements enumerated in the Release 2.1 Requirements document [DRMREQ-v2.1].

Note: This specification relies on the existence of a Public Key Infrastructure (PKI) facilitating certain security services. With a few exceptions, it is however out of scope for this specification to define the specifics of such a PKI.

2. References

2.1 Normative References

[3GPP TS 31.102] Technical Specification Group Terminals; Characteristics of the USIM Application (Release 5).

[3GPP TS 51.011] Specification of the Subscriber Identity Module –Mobile Equipment (SIM – ME) interface (Release 5).
URL:ftp://ftp.3gpp.org/specs/latest/Rel-4/51_series/

[3GPP2 C.S0023-B] URL:http://www.3gpp2.org/Public_html/specs/C.S0023-B_v1.0_040426.pdf

[AES] NIST FIPS 197: Advanced Encryption Standard (AES). November 2001.
URL:http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[AES-WRAP] Advanced Encryption Standard (AES) Key Wrap Algorithm. RFC 3394, J. Schaad and R. Housley, September 2002.
URL:http://www.ietf.org/rfc/rfc3394.txt

[Bluetooth SDP] Assigned Numbers – Service Discovery Protocol (SDP), Bluetooth SIG, August 2003.

[CertProf] “Certificate and CRL Profiles”, OMA-Security-CertProf-v1_1, Open Mobile Alliance,
URL:http://www.openmobilealliance.org

[DRM] “Digital Rights Management”, Open Mobile Alliance™, OMA-Download-DRM-v1_0,
URL:http://www.openmobilealliance.org

[DRMARCH] DRM Architecture Specification, Open Mobile Alliance™, OMA-Download_DRMARCH_v1_0,
URL:http://www.openmobilealliance.org

[DRMARCH-v2.1] ”DRM Architecture V2.0”, Open Mobile Alliance™, OMA-AD-DRM-V2-1,
URL:http://www.openmobilealliance.org

[DRMDCF] “DRM Content Format”, Open Mobile Alliance™, OMA-Download-DRMCF-v1_0,
URL:http://www.openmobilealliance.org

[DRMDCF-v2.0] “DRM Content Format V2.0”, Open Mobile Alliance™, OMA-TS-DRM-DCF-V2_0,
URL:http://www.openmobilealliance.org

[DRMDCF-v2.1] “DRM Content Format V2,1”, Open Mobile Alliance™, OMA-TS-DRM-DCF-V2_1,
URL:http://www.openmobilealliance.org

[DRMERELD-v2.1] "Enabler Release Definition for DRM V2.1". Open Mobile Alliance™. OMA-DRM-ERELD-V2_1,
URL:http://www.openmobilealliance.org

[DRMREL] “DRM Rights Expression Language”, Open Mobile Alliance™, OMA-Download-DRMREL-v1_0,
URL:http://www.openmobilealliance.org

[DRMREL-v2.0] “DRM Rights Expression Language V2.0”, Open Mobile Alliance™, OMA-TS-DRM-REL-V2_0,
URL:http://www.openmobilealliance.org

[DRMREL-v2.1] “DRM Rights Expression Language V2.1”, Open Mobile Alliance™, OMA-TS-DRM-REL-V2_1,
URL:http://www.openmobilealliance.org

[DRMREQ-v2.1] “DRM Requirements Specification V2.0”, Open Mobile Alliance™, OMA-RD-DRM-V2_1,
URL:http://www.openmobilealliance.org

[DRMROAPXSD-v2.1] “DRM ROAP schema V2.1”, Open Mobile Alliance™, OMA-TS-DRM-ROAP-V2_1,
URL:http://www.openmobilealliance.org

[DRM-v2.0] “Digital Rights Management V2.0”, Open Mobile Alliance™, OMA-TS-DRM-DRM-V2_0, URL:http://www.openmobilealliance.org

[DRM-v2.1] “Digital Rights Management V2.1”, Open Mobile Alliance™, OMA-TS-DRM-DRM-V2_1, URL:http://www.openmobilealliance.org

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

[HTTP] RFC 2616. Hypertext Transfer Protocol – HTTP/1.1. J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. June 1999, URL:http://www.ietf.org/rfc/rfc2616.txt

[IOPPROC] "OMA Interoperability Policy and Process", Version 1.1, Open Mobile Alliance™, OMA-IOP-Process-V1_1, URL:http://www.openmobilealliance.org/

[ISO/IEC 18033] ISO/IEC 18033-2, Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers. CD3, January 2004.

[OBEX] IrDA Object Exchange Protocol (OBEX), Version 1.3, January 2003.

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

[OCSP-MP] OMA Online Certificate Status Protocol (profile of [OCSP]) V 1.0, Open Mobile Alliance™, URL:http://www.openmobilealliance.org/

[ODRL] “Open Digital Rights Language (ODRL)”, Version 1.1, 8 August 2002, Open Mobile Alliance™ URL:http://odrl.net/1.1/ODRL-11.pdf or URL:http://www.w3.org/TR/odrl/

[PKCS-1] “PKCS #1 v2.1: RSA Cryptography Standard”, RSA Laboratories. June 2002. URL:http://www.rsasecurity.com/rsalabs

[RFC 2616] “Hypertext Transfer Protocol -- HTTP/1.1”, R. Fielding, et al, June 1999, URL:http://www.ietf.org/rfc/rfc2616.txt

[RFC 2965] “HTTP State Management Mechanism”. D. Kristol, L. Montulli, October 2000 URL:http://www.ietf.org/rfc/rfc2965.txt.

[RFC2045] “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

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

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

[RFC2387] “The MIME Multipart/Related Content-type”, E. Levinson, 1998, URL:http://www.ietf.org/

[RFC2396] “Uniform Resource Identifiers (URI): Generic Syntax”. T. Berners-Lee, R. Fielding, L. Masinter. August 1998, URL:http://www.ietf.org/rfc/rfc2396.txt

[RFC2630] “Cryptographic Message Syntax”, R. Housley, June 1999, URL:http://www.ietf.org/rfc/rfc2630.txt

[RFC3280] Housley, R., Polk, W., Ford, W. and D. Solo, "Internet Public Key Infrastructure - Certificate and Certificate Revocation List (CRL) Profile", April 2002. URL:http://www.ietf.org/rfc/rfc3280.txt

[RFC3546] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, T. Wright, “Transport Layer Security (TLS) Extensions”. June 2003.
URL:http://www.ietf.org/rfc/rfc3546.txt

[RFC4234] D. Crocker, Ed., P. Overell. “Augmented BNF for Syntax Specifications: ABNF.” October 2005.

URL:http://www.ietf.org/rfc/rfc4234.txt

[SHA-1] NIST FIPS 180-2: Secure Hash Standard. August 2002.
URL:http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

[UAProf] “User Agent Profile”, OMA -UAProf-v2_0, Open Mobile Alliance™,
URL:http://www.openmobilealliance.org

[WBXML] “Binary XML Content Format Specification”. WAP Forum™. WAP-192-WBXML.
URL: http://www.openmobilealliance.org

[WIM] “Wireless Identity Module Version 1.1. Part: Security”, OMA-WAP-WIM-v1_1, Open Mobile Alliance™,
URL:http://www.openmobilealliance.org

[X9.42] ANSI X9.42 Public Key Cryptography For The Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography, 2003.

[X9.44] Draft ANSI X9.44, Public Key Cryptography for the Financial Services Industry – Key Establishment Using Integer Factorization Cryptography. Draft 6, 2003.

[X9.63] ANSI X9.63 Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography, 2001.

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

[XML-DSIG] XML-Signature Syntax and Processing. D. Eastlake, J. Reagle, and D. Solo. W3C Recommendation, February 2002.
URL:http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

[XML-Enc] XML Encryption Syntax and Processing. D. Eastlake and J. Reagle. W3C Recommendation, December 2002.
URL:http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/

[XML-Schema] XML Schema Part 1: Structures D. Beech, M. Maloney, and N. Mendelsohn. W3C Recommendation, May 2001.
URL:http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

XML Schema Part 2: Datatypes. P. Biron and A. Malhotra. W3C Recommendation, May 2001,
URL:http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

2.2 Informative References

[3GPP PSS] Transparent end-to-end packet switched streaming service (PSS); 3GPP TS-26.234; Protocols and codecs – Release 6,
URL:http://www.3gpp.org/

[DDOS] "Recommendations for the Protection against Distributed Denial-of-Service Attacks in the Internet", Bundesamt für die Sicherheit in der Informationstechnik, 2000.
URL:http://www.iwar.org.uk/comsec/resources/dos/ddos_en.htm

[DLOTA] “OMA Download version 1.0.” Open Mobile Alliance™. OMA-Download-OTA-V1_0.
URL:http://www.openmobilealliance.org

[DLOTA20] “OMA Download version 2.0.” Open Mobile Alliance™. OMA-Download-OTA-V2_0
URL:http://www.openmobilealliance.org

[DRMARCH-v2.0] “DRM Architecture V2.0”, Open Mobile Alliance™. OMA-AD-DRM-V2_0
URL:http://www.openmobilealliance.org

[MMSENC] “Multimedia Messaging Service, Encapsulation Protocol”, Open Mobile Alliance™. OMA-TS-MMS- ENC-V1_3.
URL:http://www.openmobilealliance.org

[MTP] Media Transfer Protocol v.1.0 Spec and MTP v.1.0 Adopters Agreement, URL: http://www.usb.org/developers/devclass_docs/MTP_1.0.zip
URL:http://www.usb.org/developers/devclass_docs#approved

[PUSHOTA] “Push OTA Protocol Specification.” Open Mobile Alliance™™. WAP-235-PushOTA.
URL:http://www.openmobilealliance.org

[TS26.244] “Transparent end-to-end Packet-switched Streaming Service (PSS); File Format”, Version 1.2.0, The Third Generation Partnership Project, TS-26.244,
URL:http://www.3gpp.org/

[UICC] “Smart cards; UICC-Terminal interface; Physical and logical characteristics (release 5)”, ETSI 102.221 ,
URL:http://www.etsi.org

[WSP] "Wireless Session Protocol Specification” Open Mobile Alliance™™. WAP-230-WSP.
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 specification uses schema documents conforming to W3C XML Schema [XML-Schema] and normative text to describe the syntax and semantics of XML-encoded ROAP messages.

Listing of Rights Object Acquisition Protocol (ROAP) schemas appear like this.

The following typographical conventions are used in the body of the text: **<XML Element>**, ***XMLAttribute***, **XMLType**, `ASN.1ValueOrType`.

3.2 Definitions

Backup/Remote Storage Transferring Rights Objects and Content Objects to another location with the intention of transferring them back to the original Device.

Billing Service Provider The entity responsible for collecting payment from a User.

Combined Delivery A Release 1 method for delivering DRM Content and Rights Object. The Rights Object and DRM Content are delivered together in a single entity, the DRM Message.

Composite Object A content object that contains one or more Media Objects by means of inclusion.

Confidentiality The property that information is not made available or disclosed to unauthorised individuals, entities or processes. (From [ISO 7498-2])

Connected Device A Connected Device is a Device that is capable of directly connecting to a Rights Issuer using an appropriate protocol over an appropriate transport/network layer interface. E,g, HTTP over TCP-IP.

Content One or more Media Objects

Content Issuer The entity making content available to the DRM Agent in a Device.

Content Provider An entity that is either a Content Issuer or a Rights Issuer.

Content subscription A subscription that a User has with a Content Provider for the purposes of paying for DRM Content purchased from that Content Provider and played on a Users Device.

Device A Device is the entity (hardware/software or combination thereof) within a user-equipment that implements a DRM Agent. The Device is also conformant to the OMA DRM specifications.

In the case where functionality is specific to either Connected Devices or Unconnected Devices the explicit terminology (i.e. Unconnected Device or Connected Device) will be used, in all other cases the term Device generically applies to both Connected Devices and Unconnected Devices.

Device Revocation The process of an RI indicating that a Device is no longer trusted to acquire ROs.

Device Rights Object An RO dedicated for a particular Device by means of the Device Public Key.

Disabled Rights Object A Rights Object that is indicated as unusable.

Domain A set of Devices, which are able to share Domain Rights Objects. Devices in a Domain share a Domain Key. A Domain is defined and managed by an RI.

Domain baseID The first (leading) characters that precede the Domain Generation Counter in the Domain Identifier.

Domain Consume Expiry Time An absolute time after which the Device is not allowed to consume ROs for this Domain.

Domain Context The Domain Context consists of information necessary for the Device to install Domain Rights Objects, such as Domain Key, Domain Identifier and Expiry Time.

Domain Context Expiry Time An absolute time after which the Device is not allowed to install ROs for this Domain. Usage of ROs installed before the expiry time are not affected by the expiry.

Domain Generation A Counter reflecting the number of times the Domain has been upgraded. The Domain Generation is a part of the Domain Identifier (the last three digits).

Domain Identifier A unique string identifier of the Domain Key

Domain Key A 128 bit symmetric cipher key

Domain Revocation The process of an RI indicating that a Domain Key is not trusted for protection of Domain ROs.

Domain Rights Object An RO that is dedicated to Devices in a particular Domain by means of a Domain Key.

DRM Agent The entity in the Device that manages Permissions for Media Objects on the Device.

DRM Content Media Objects that are consumed according to a set of Permissions in a Rights Object.

DRM Message An OMA DRM Release 1 term defined in [DRM]

DRM Time A secure, non user-changeable time source. The DRM Time is measured in the UTC time scale.

Enabled Rights Object A Rights Object that is indicated as usable.

Forward Lock An OMA DRM Release 1 term defined in [DRM]

Hash Chains A Method of derivation of Domain Keys of different Domain Generations.

Integrity The property that data has not been altered or destroyed in an unauthorised manner. (ISO 7498-2)

Join Domain The process of an RI including a Device in a Domain.

Leave (De-Join) Domain The process of an RI excluding a non-revoked Device from a Domain.

Media Object A digital work e.g. a ringing tone, a screen saver, a Java game or a Composite Objec*t*.

Metered Content When DRM Content is accessed via an RO that contains the **<tracked>** element, the content is said to be Metered Content.

Metering Recording of usage information (the number of times that the associated DRM Content has been consumed and the accumulated usage time) upon consumption of an RO which contains the <tracked> element. Metering enables a Rights Issuer to collect usage information from Devices for the purpose of royalty collection.

Metering Information The information that is recorded for purposes of Metering

Metering Report A generic term used to refer to the report containing aggregated Metering Information. This may be used to refer to the Raw Metering Report or the XML encoded content of the **<meteringReport>** element of a ROAP-MeteringReportSubmit message.

Permission Actual usages or activities allowed (by the Rights Issuer) over DRM Content (From [ODRL]

Play To create a transient, perceivable rendition of a resource (From [MPEG21 RDD])

Raw Metering Report Metering Information in the ABNF format as defined in section 11.4.

Restore Transferring the DRM Content and/or Rights Objects from an external location back to the Device from which they were backed up.

Revoke Process of declaring a Device or Rights Issuer certificate as invalid.

RI Context RI Context (Rights Issuer Context) consists of information that was negotiated with a given Rights Issuer, during the 4-pass Registration Protocol such as RI ID, RI certificate chain, version, algorithms and other information. This RI Context is necessary for a Device to successfully participate in all the protocols of the ROAP suite, except the Registration Protocol.

Rights Issuer An entity that issues Rights Objects to OMA DRM Conformant Devices.

Rights Object A collection of Permissions and other attributes which are linked to DRM Content.

Rights Object Acquisition Protocol (ROAP) A protocol defined within this specification. This protocol enables Devices to request and acquire Rights Objects from a Rights Issuer.

Rights Object Installation Confirmation Installation Confirmation occurs when the Device informs the RI from whom it received ROs of the success or failure of its attempts to install those ROs.

ROAP Trigger An XML document including a URL that, when received by the Device, initiates the ROAP.

ROAP URL A URL according to [RFC2396] that is specifically used by a Device for exchanging ROAP PDU's with a Rights Issuer.

Separate Delivery A Release 1 term defined in DRM

Stateful Rights Stateful Rights are Rights Objects for which the Device has to explicitly maintain state information, so that the constraints and permissions expressed in the RO can be enforced correctly. An RO containing any of the following constraints is considered Stateful Rights: <interval>, <count>, <timed-count>, or <accumulated>. Additionally an RO with <export> permission and mode attribute of "move" is Stateful Rights.

Stateless Rights Stateless Rights are Rights Objects for which the Device does not have to maintain state information.

Superdistribution A mechanism that (1) allows a User to distribute DRM Content to other Devices through potentially insecure channels and (2) enables the User of that Device to obtain a Rights Object for the superdistributed DRM Content.

Unconnected Device An Unconnected Device is a Device that is capable of connecting to a Rights Issuer via a Connected Device using an appropriate protocol over a local connectivity technology. E.g. OBEX over IrDA, Bluetooth or USB. An Unconnected Device may support DRM Time.

User The human user of a Device. The User does not necessarily own the Device.

Well-intentioned Attempt A "well-intentioned attempt" means that a Device attempted to send a message under circumstances where the network connection is known (to the extent possible) to be present. If there is no network connection present then an attempt to send a message should not be regarded as well-intentioned.

3.3 Abbreviations

3GPP	3rd Generation Partnership Project
3GPP PSS	3rd Generation Partnership Project Packet-switched Streaming Service
CA	Certification Authority
CBC	Cipher Block Chaining
CEK	Content Encryption Key
CI	Content Issuer
DCF	DRM Content Format
DD	Download Descriptor
DER	Distinguished Encoding Rules
DRM	Digital Rights Management
GUID	Globally Unique Identifier
HTTP	HyperText Transfer Protocol
IMSI	International Mobile Subscriber Identity
ISO	International Standards Organisation
LAN	Local Area Network
MAC	Message Authentication Code
ME	Mobile Equipment
MMS	Multimedia Messaging Service
MPEG	Moving Picture Expert Group
OCSP	Online Certificate Status Protocol
OMA	Open Mobile Alliance
OMNA	Open Mobile Naming Authority (see http://www.openmobilealliance.org/tech/omna/index.htm)
OTA	Over The Air (i.e. transfer over a wireless connection)
PC	Personal Computer

PDA	Personal Digital Assistant
PDCF	Packetised DRM Content Format
PDU	Protocol Data Unit
PKC	Public Key Certificate
PKC-ID	PKC Identifier: the hash of the Public Key Certificate
PKI	Public Key Infrastructure
PSS	Packet-Switched Streaming Service (see [3GPP PSS])
REK	Rights Object Encryption Key
REL	Rights Expression Language
RFC	Request For Comments
RI	Rights Issuer
RO	Rights Object
ROAP	Rights Object Acquisition Protocol
RSA	Rivest-Shamir-Adelman public key algorithm
RSA-PSS	RSA Probabilistic Signature Scheme (see [PKCS-1])
SCR	Static Conformance Requirement
SHA-1	Secure Hash Algorithm
SIM	Subscriber Identity Module
SMIL	Synchronised Multimedia Integration Language
SMS	Short Messaging Service
TLS	Transport Layer Security
UA	User Agent
URI	Uniform Resource Indicator
URL	Uniform Resource Locator
USIM	Universal Subscriber Identity Module
UTC	Coordinated Universal Time
WIM	Wireless Identity Module
WLAN	Wireless Local Area Network

4. Introduction

OMA "Digital Rights Management" (DRM) enables the distribution and consumption of digital content in a controlled manner by enabling

Content Issuers to distribute DRM Content and Rights Issuers to issue Rights Objects for the DRM Content.

The OMA DRM system is independent of media object formats, operating systems, and runtime environments. Content protected by the DRM can be of a wide variety: games, ring tones, photos, music clips, video clips, streaming media, etc. A content provider can grant appropriate permissions to the user for each of these media objects.

The DRM Content can be delivered to the Device by any means (over the air, LAN/WLAN, local connectivity, removable media, etc.). It is possible to deliver DRM Content and the associated Rights Object together, but it is also possible to send them separately. The system does not imply any order or "bundling" of these two objects.

It is not within the scope of the DRM system to address the specific payment methods employed by the Rights Issuers.

This specification is one part of a set of specifications developed by OMA to address the need for digital rights management. For a detailed discussion of the overall system architecture, please refer to [DRMARCH-v2.1]. For a detailed discussion of the Rights Expression Language that is used to construct the Rights Objects, please refer to [DRMREL-v2.1]The DRM Content Format is specified in the [DRMDCF-v2.1] specification.

This specification defines an end-to-end system for DRM Content distribution. Section 5 specifies a Rights Object Acquisition Protocol (ROAP). Section 7 describes the key management schemes utilised in this specification. Section 8 describes the Domains functionality – sharing of content and rights among a set of Devices enrolled into a Domain. Section 10 through 19 deals with various other aspects of this system: super distribution, transport mappings for ROAP, etc. Finally, the appendices describe related normative as well as informative topics.

4.1 Version 1.0

The OMA DRM v1.0 specification provides some fundamental building blocks for a DRM system without addressing the complete security necessary for a robust, end-to-end DRM system that takes into account the need for secure distribution, authentication of Devices, revocation and other aspects.

The most important OMA DRM v1.0 functionality is listed below:

- ✓ Forward Lock to prevent forwarding of content if delivered in a DRM Message;
- ✓ Support for the "Content-Transfer-Encoding" and "Content-ID" headers in the DRM Message;
- ✓ Support for the combined delivery of rights and content;
- ✓ Support for separate delivery of rights and content, including superdistribution of said content;
- ✓ Control of content usage based on the specified rights and constraints.

4.2 Version 2.0

The main differences between OMA DRM v1.0 and OMA DRM v2.0 are significantly improved security and functionality. Improved security is for example achieved by providing bilateral authorization between Rights Issuer and Device, based on PKI certificates and by confidentiality and integrity protecting Rights Objects. Improved functionality and usability is for example achieved by providing preview functions, mechanisms for sharing of content within a registered community of devices, called a domain, and by enabling devices without a wide-area network connection (unconnected devices) to participate in the system, and consume DRM Content.

The OMA DRM v2.0 specification enables content providers to grant permissions for media objects that define how they should be consumed. A content provider can grant appropriate permissions to the user for each of these media objects. A Rights Object is bound to a Device, by protecting it with the device public key, or to small domains of devices, by protecting it with a domain key. The content is distributed with cryptographic protection; hence, the DRM Content is not usable without the associated Rights Object on a Device. Given this fact, fundamentally, the users are purchasing permissions embodied in Rights Objects and the Rights Objects need to be handled in a secure and un-compromising manner.

4.2.1 Version 2.0.1

The most important DRM changes introduced in DRM v2.0.1 compared with DRM v2.0 are summarized in section 4.2.1 of the DRM Specification [DRM-v2.1] in the DRM 2.0.1 ERP. These changes are considered to require special consideration in implementation. Many of the identified changes are bug fixes which if not implemented correctly may result in interoperability problems between conformant and non-conformant devices. Companies with existing DRM 2.0 implementations should take careful consideration of these changes.

4.3 Version 2.1

OMA DRM v2.1 has been developed as a result of market feedback. The main differences between OMA DRM v2.0 and OMA DRM v2.1 are the addition of several features on top of OMA DRM v2.0, including:

- Metering, primarily intended for information gathering. By means of metering, actual content usage information can be provided to Rights Issuers, thereby enabling royalty collection based on actual usage of content.
- Content differentiation, defining a mechanism to control how content can be consumed. For example, this mechanism can prevent that a music track is used as ringtone.
- RO installation confirmation.
- Additional metadata, such as artist, title and genre
- Support for user editable metadata, in addition to content issuer defined metadata.
- A binary format for ROAP triggers to improve communication efficiency.
- New domain property (noConsumeAfter) to simplify “temporary sharing” business models
- RO upload functionality to enable users to upload Rights from their old device to a Rights Issuer so that these Rights can be downloaded to their new device.
- Improved extensibility for future versions.

The DRM 2.1 features have minimum impact on the DRM 2.0 architecture and are defined in a DRM 2.0 backward compatible manner.

5. The Rights Object Acquisition Protocol (ROAP) Suite

5.1 Overview

The Rights Object Acquisition Protocol (ROAP) is the common name for a suite of DRM security protocols between a Rights Issuer (RI) and a DRM Agent in a Device. The protocol suite contains a 4-pass protocol for registration of a Device with an RI, a simple 2-pass variant of this to allow an RI and Device to exchange DRM IDs and two protocols by which the Device requests and acquires Rights Objects (RO) and one protocol by which the Device may request to upload ROs.The 2-pass RO acquisition protocol encompasses request and delivery of an RO whereas the 1-pass RO acquisition protocol is only a delivery of an RO from an RI to a Device (e.g. messaging/push). The RO acquisition protocols can be optionally extended to allow a Device to confirm to the RI whether it has successfully installed the ROs delivered. The ROAP suite also includes 2-pass protocols for Devices joining and leaving a Domain; the Join Domain protocol and the Leave Domain protocol. The ROAP suite also includes a 2-pass protocol for Devices to submit Metering Reports to RIs for the purpose of royalty collection.

For RIs, execution of a ROAP protocol may involve interaction with one or more OCSP responders, in order to retrieve a valid set of OCSP responses. This interaction is not always needed, and is illustrated in the following flow diagrams with dotted lined.

5.1.1 The 4-pass Registration Protocol

The Registration protocol is a complete security information exchange and handshake between the RI and the Device and is generally only executed at first contact, but may also be executed when there is a need to update the exchanged security information, or when DRM Time in the Device is deemed inaccurate by the Rights Issuer. This protocol includes negotiation of protocol parameters and protocol version, cryptographic algorithms, exchange of certificate preferences, optional exchange of certificates, mutual authentication of Device and RI, integrity protection of protocol messages and optional Device DRM Time synchronisation.

Successful completion of the Registration protocol results in the establishment of an RI Context in the Device containing RI-specific security related information such as agreed protocol parameters, protocol version, and certificate preferences. An RI Context is necessary for execution of the other protocols in the ROAP suite.

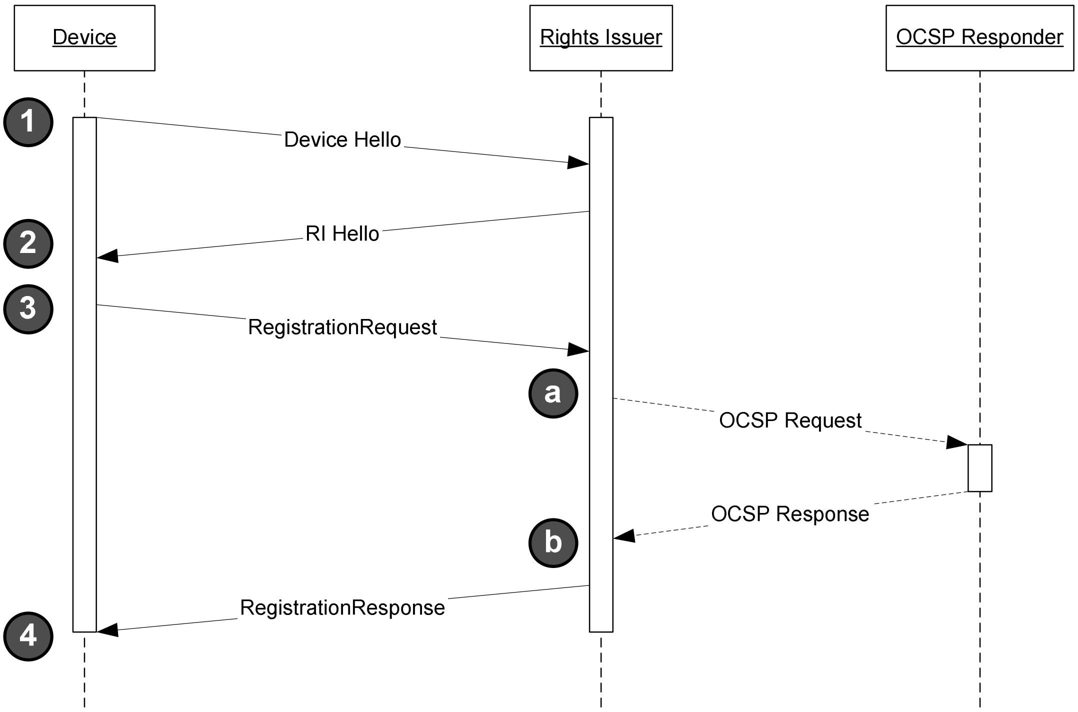

Figure 1: The 4-pass Registration Protocol

As indicated in the figure above, the RI may optionally perform a nonce-based OCSP request for its own certificate (using a nonce supplied by the Device) during the registration protocol, and then provide the Device with the returned OCSP response. The RI will perform this nonce-based OCSP request if it determines that the Device's DRM Time is inaccurate. A Device will then be able to adjust its DRM Time based on the time in the OCSP response. The Device MUST maintain one DRM Time per trust model. During ROAP communication and Rights Object evaluation, the Device needs to select the correct DRM Time depending on the trust model it is currently working with. Further usages of the term "DRM Time" imply this. If the Device is an Unconnected Device that does not support DRM Time, the RI must always perform a nonce-based OCSP request for its own certificate (using a nonce supplied by the Device) during the registration protocol.

5.1.2 The 2-pass Identification Protocol

The 2-pass Device Identification Protocol allows an RI and Device to exchange DRM IDs. This variant of the protocol can be used by Rights Issuers and Content Issuers to determine the DRM ID of a Device in order to customise presentation logic and therefore improve the user experience. This variant of the protocol does not include any mutual authentication or integrity protection and therefore the DRM IDs exchanged as part of the protocol SHOULD be verified at a later point in time using one of the other variants of the protocol.

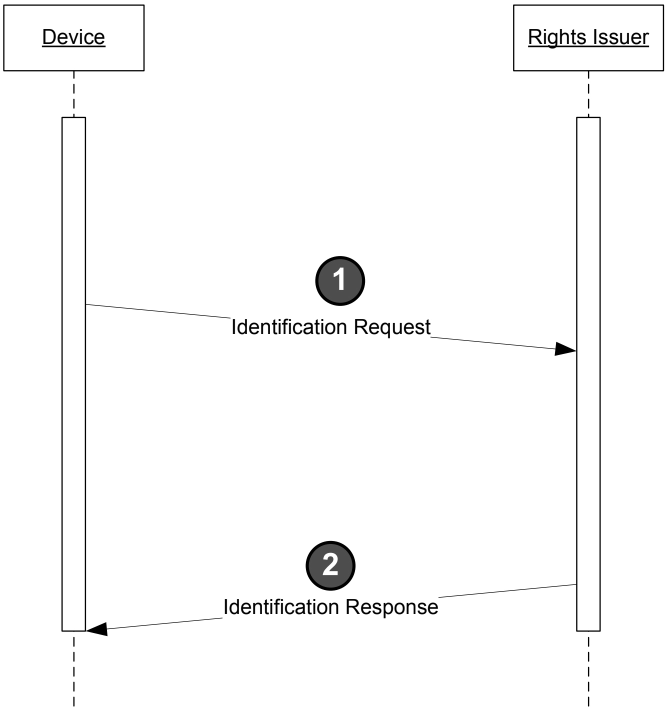

Figure 2: The 2-pass Identification Protocol

5.1.3 The 2-pass Rights Object Acquisition Protocol

The 2-pass RO acquisition protocol is the protocol by which the Device acquires Rights Objects. This protocol includes mutual authentication of Device and RI, integrity-protected request and delivery of ROs, and the secure transfer of cryptographic keying material necessary to process the RO. The successful execution of this protocol assumes the Device to have a pre-established RI Context with the RI.

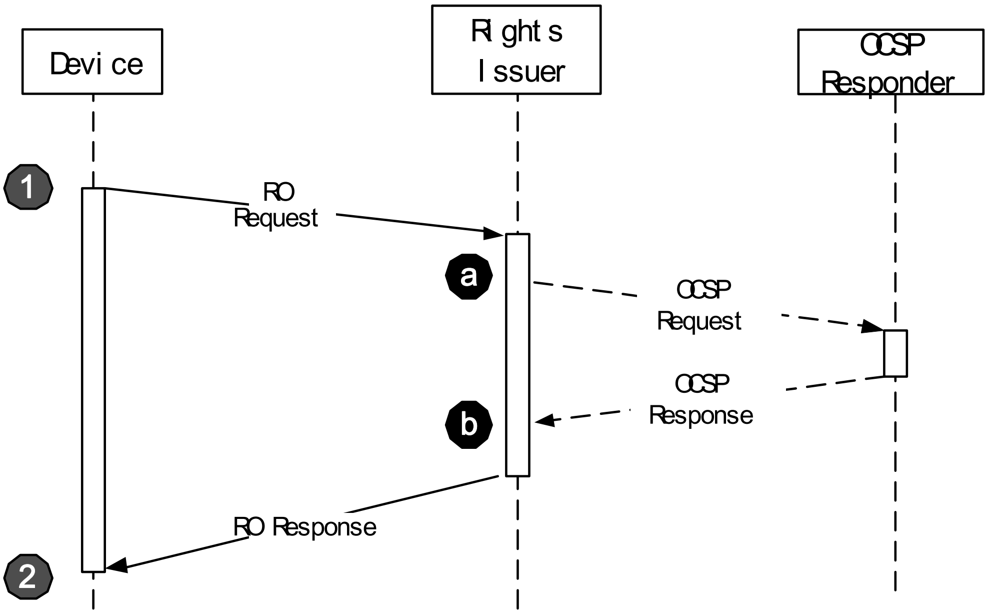

Figure 3: The 2-pass Rights Object Acquisition Protocol

5.1.4 The 4-pass Confirmed Rights Object Acquisition Protocol

If the RI requires the Device to confirm the installation of ROs delivered via the 2-pass RO acquisition protocol, then the Rights Object Acquisition protocol is extended with two additional messages which allow the installation status to be communicated the RI as shown in Figure 4. This extension is optional. The successful execution of this protocol assumes the Device to have a pre-established RI Context with the RI.

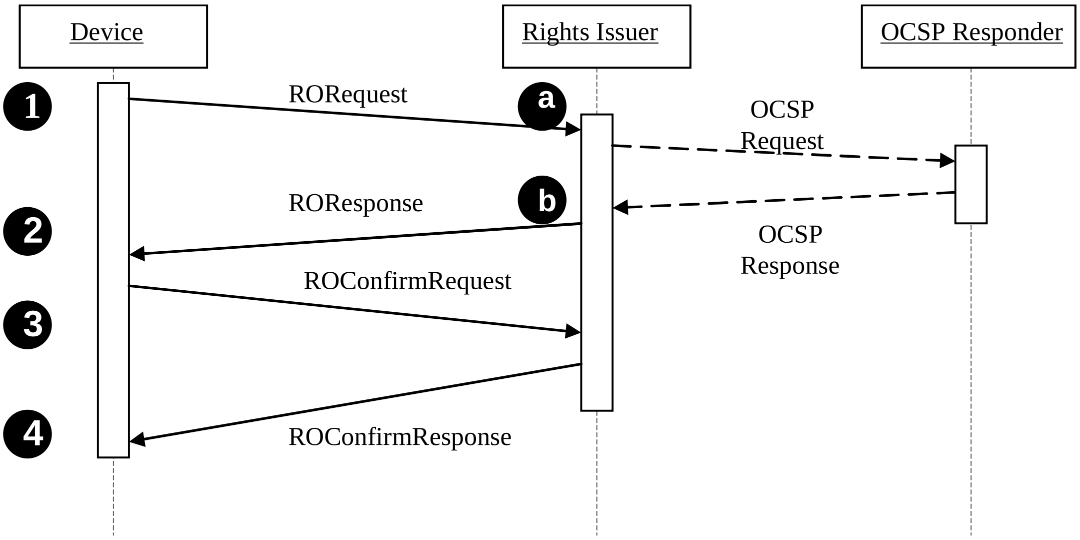

Figure 4: 4-pass confirmed Rights Object Acquisition Protocol

5.1.5 The 1-pass Rights Object Acquisition Protocol

The 1-pass RO acquisition protocol is designed to meet the messaging/push use case. Its successful execution assumes the Device to have an existing RI Context with the sending RI. In contrast to the 2-pass RO acquisition protocol, it is initiated unilaterally by the RI and requires no messages to be sent by the Device. One use case is distribution of Rights Objects at regular intervals, e.g. supporting a content subscription. The 1-pass protocol is essentially the last message of the 2-pass variant.

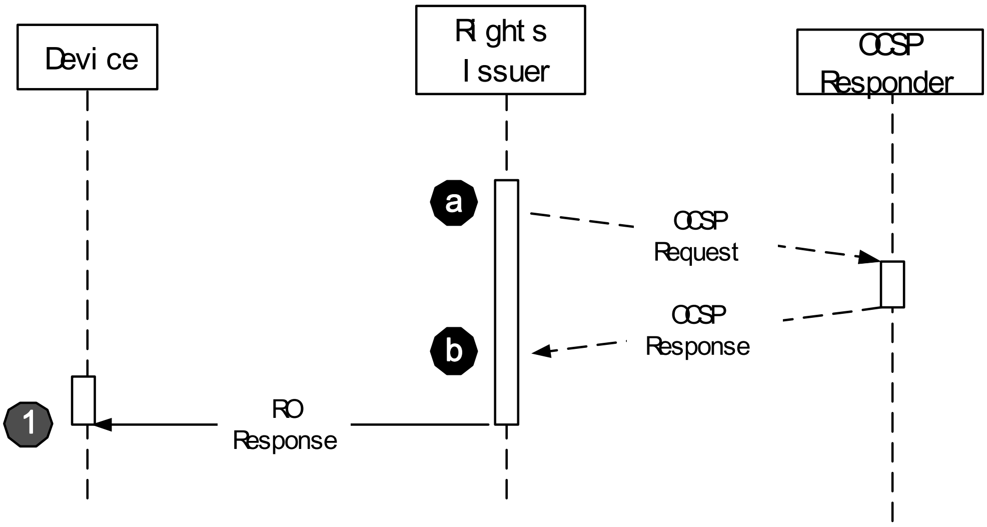

Figure 5: The 1-pass Rights Object Acquisition Protocol

5.1.6 The 3-pass Confirmed Rights Object Acquisition Protocol

If the RI requires the Device to confirm the installation of ROs delivered via the 1-pass RO acquisition protocol, then the Rights Object Acquisition protocol is extended with two additional messages which allow the installation status to be communicated the RI as shown in Figure 6. This extension is optional. The successful execution of this protocol assumes the Device to have a pre-established RI Context with the RI.

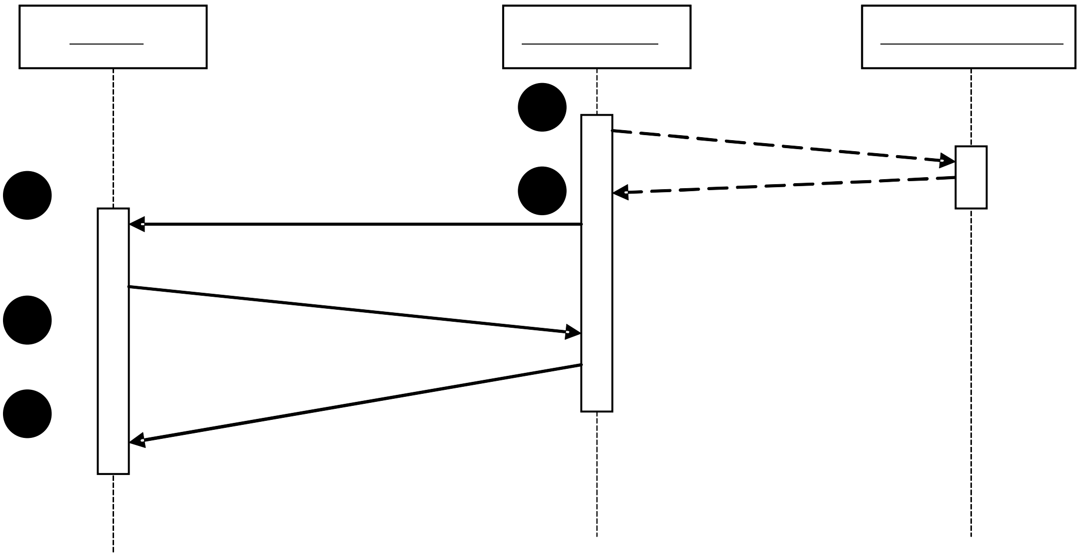

Figure 6: 3-pass Confirmed RO Acquisition Protocol

5.1.7 The 2-pass Join Domain Protocol

The Join Domain protocol is the protocol by which a Device joins a Domain. The protocol assumes an existing RI Context with the RI administering the Domain.

Successful completion of the Join Domain protocol results in the establishment of a Domain Context in the Device containing Domain-specific security related information including a Domain Key. A Domain Context is necessary for the Device to be able to install and utilize Domain ROs.

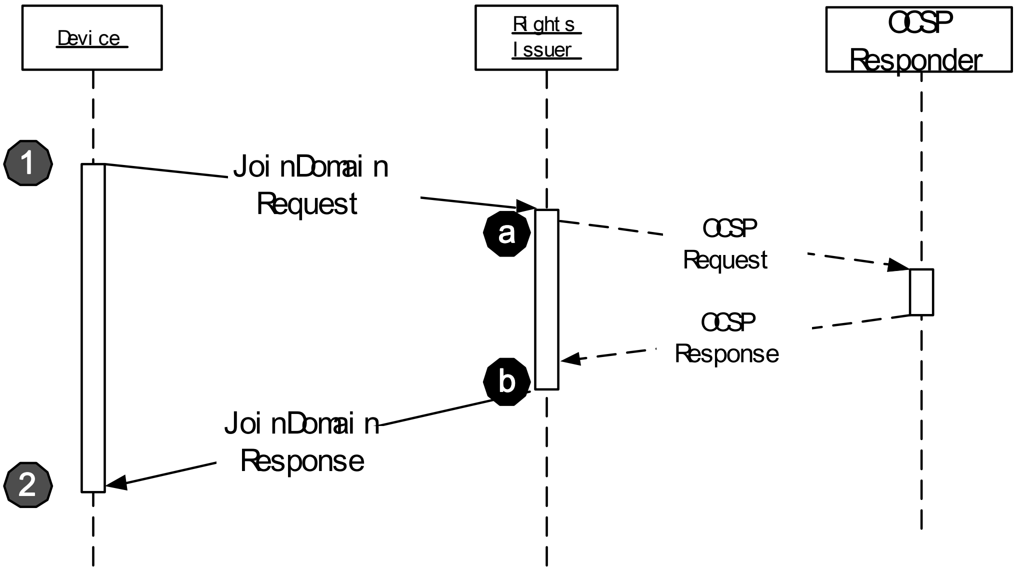

Figure 7: The 2-pass Join Domain Protocol

5.1.8 The 2-pass Leave Domain Protocol

The Leave Domain protocol is the protocol by which a Device leaves a Domain. The protocol assumes an existing RI Context with the RI administering the Domain.

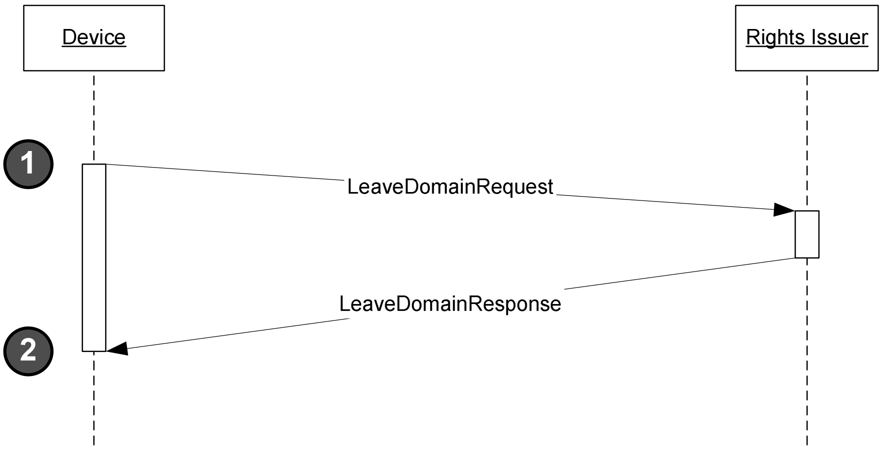

Figure 8: The 2-pass Leave Domain Protocol

5.1.9 The 2-pass Metering Report Protocol

The Metering Report protocol is the protocol by which a Device submits Metering Reports to the RI. The protocol assumes an existing RI Context with the RI that issued the ROs for which the associated Metering Information will be reported.

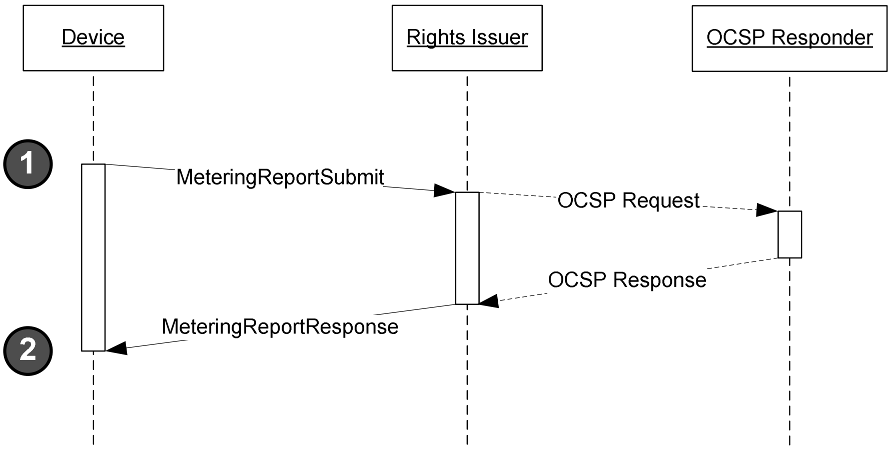

Figure 9: The 2-pass Metering Report Protocol

5.1.10 The 2-pass Rights Object Upload Protocol

The Rights Object Upload protocol is the protocol by which a Device submits a Rights Object Upload Request to the RI. Rights Objects MAY only be uploaded to the same RI that originally issued the RO. The protocol assumes the Device has a valid RI context for the associated RI.

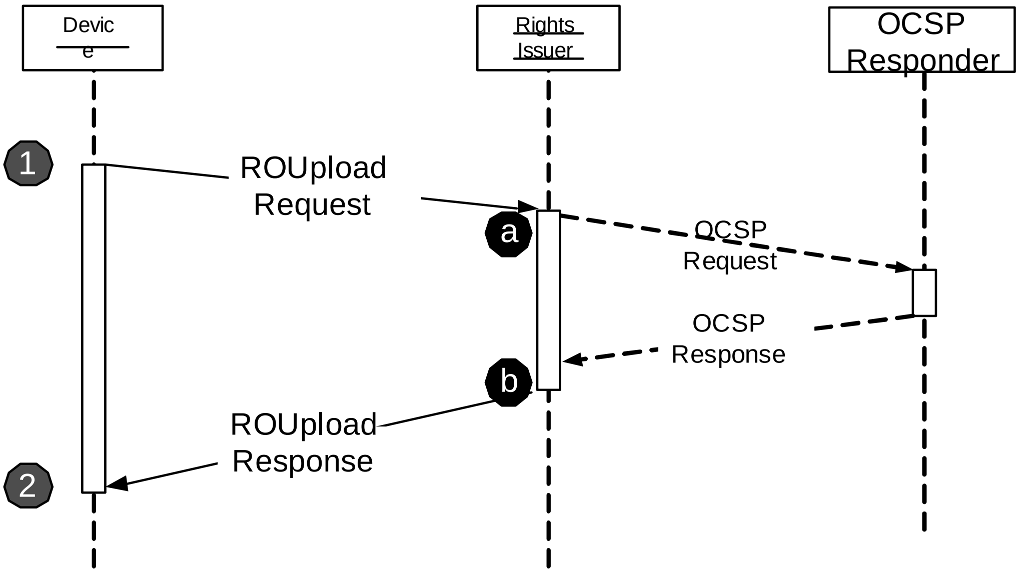

Figure 10: The 2-pass Rights Object Upload Protocol

5.1.11 The ROAP Trigger

All protocols included in the ROAP suite except the 1-pass RO acquisition protocol may be initiated using a ROAP Trigger. The Device MAY also initiate them unilaterally as a result of user interactions. The Rights Issuer generates and sends the ROAP Trigger to the Device to trigger a ROAP protocol exchange. Alternatively, the Rights Issuer may delegate ROAP Trigger generation to other systems by providing necessary information (such as Rights Object identifiers and Domain identifiers) to these systems. A ROAP Trigger (whether generated directly or indirectly by the RI) may also be transmitted to the Device by other systems (e.g. by a Content Issuer).

When the Device receives the ROAP Trigger, it initiates the ROAP protocol exchange as soon as possible. Prior to initiating a ROAP protocol exchange a Device may have to obtain user consent, section 5.1.13 defines when explicit user consent is required.Since the ROAP comprises several protocols, the ROAP Trigger includes an indication of the actual protocol (Registration, Identification, RO acquisition, Join Domain, Leave Domain, Metering Report or RO Upload) to be started by the Device.

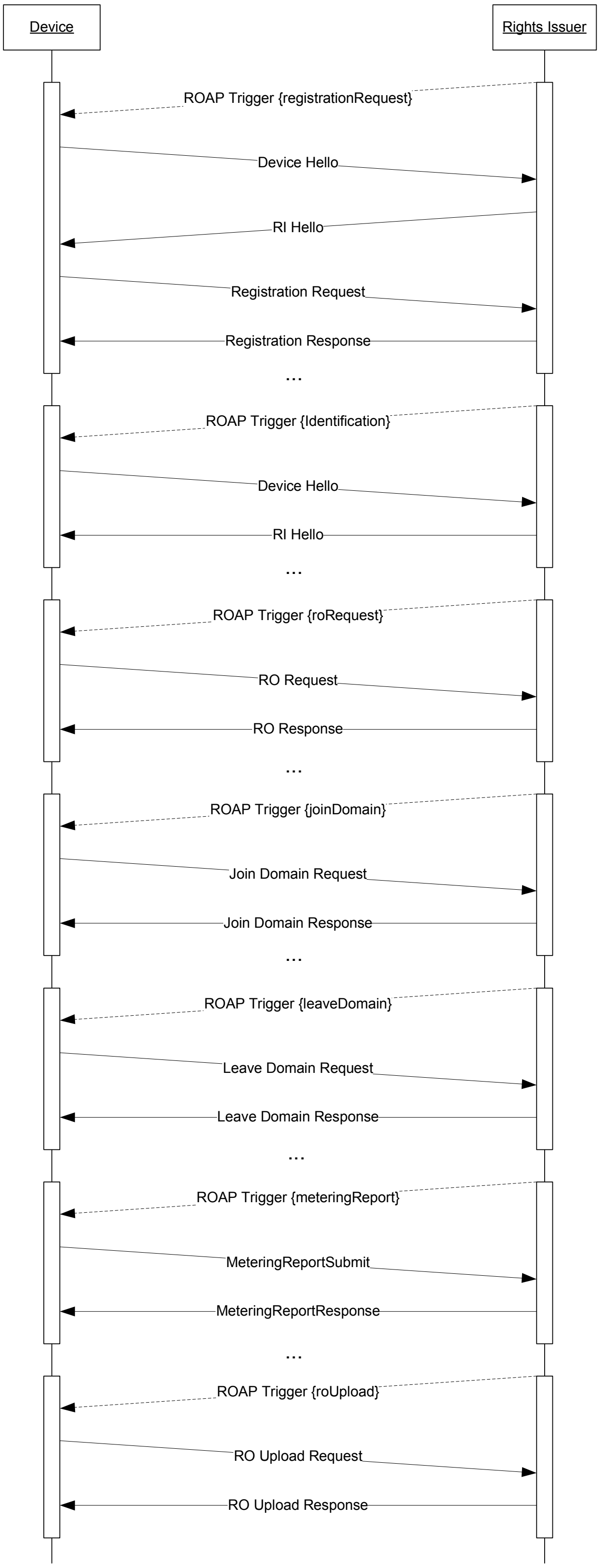
Device
Rights Issuer
ROAP Trigger {registrationRequest}
Device Hello
RI Hello
Registration Request
Registration Response
...
ROAP Trigger {Identification}
Device Hello
RI Hello
...
ROAP Trigger {roRequest}
RO Request
RO Response
...
ROAP Trigger {joinDomain}
Join Domain Request
Join Domain Response
...
ROAP Trigger {leaveDomain}
Leave Domain Request
Leave Domain Response
...
ROAP Trigger {meteringReport}
MeteringReportSubmit
MeteringReportResponse
...
ROAP Trigger {roUpload}
RO Upload Request
RO Upload Response

Figure 11: ROAP Trigger

5.1.12 ROAP URL's

The value of ROAP URLs MUST be a URL according to [RFC2396], and MUST be an absolute identifier. A Device MUST use the URL without modification, and SHOULD not try to interpret the URL in anyway except for the **scheme** and **authority** components as defined in [RFC2396].

A Rights Issuer MAY add additional arguments to the ROAP URL. Such arguments could for instance be used to match an incoming ROAP request with a transaction known to the RI. What data can be passed is outside the scope of this specification.

ROAP URL’s from ROAP triggers take precedence over other ROAP URL’s, and MUST be used for all explicitly and implicitly triggered ROAP protocols. The lifetime of the ROAP trigger is defined to end as soon as the explicitly triggered ROAP PDU is send and a Response is received. If the explicitly triggered ROAP PDU is send again during a retry, the lifetime of the ROAP trigger is extended to include the Request and Response of the explicitly triggered ROAP PDU of such retry.

Which protocols might be implicitly triggered by a ROAP trigger is summarised in the next table. See section 5.2.1.

ROAP Protocol	Possible implicit ROAP protocols
4 pass Registration	-
2 pass Identification	-
2 pass RORequest	4 pass Registration if RI context is unavailable or invalid, or in case of a DeviceTime or NotRegistered Error. 2 pass JoinDomain if Domain Context is unavailable, not of correct generation, or invalid.
1 pass ROResponse	-
2 Pass JoinDomain	4 pass Registration if RI context is unavailable or invalid, or in case of a DeviceTime or NotRegistered Error.
2 Pass LeaveDomain	4 pass Registration if RI context is unavailable or invalid, or in case of a DeviceTime Error.
2 Pass Metering Report	4 pass Registration if RI context is unavailable or invalid, or in case of a DeviceTime or NotRegistered Error
2 Pass ROUpload	4 pass Registration if RI context is invalid, or in case of a DeviceTime or NotRegistered Error

Table 1: Protocols implicitly triggered by a ROAP trigger

When a ROAP trigger is unavailable the **riURL** from the RI Context MUST be used. By using such context **riURL** a Device is able to perform unsolicited ROAP protocols to, for instance, leave a Domain.

5.1.13 Rules for Obtaining User Consent

There are various points within the execution of ROAP, the processing of DCFs and the process of installing Domain and Device ROs that a Device may have to obtain user consent. This section defines when explicit user consent is required. Some explicit user interactions may not be necessary if the Device implements a User Confirmation Whitelist that contains the Fully Qualified Domain Name of authorised RIs and the corresponding riID. Devices SHOULD implement a User Confirmation Whitelist.

- A Device MUST obtain the user's consent before initiating a request to the (P)DCF RightsIssuerURL. The DRM Agent MUST NOT attempt to acquire an RO for the DCF if the user does not provide consent.
- If the DCF includes a Silent header with a specified silent rights URL or a Preview header with method "preview-rights" and a specified preview rights URL, the DRM Agent MUST compare the domain name of the silent or preview URL with the list of authorised domain names already stored by the DRM Agent for that RI. The DRM Agent MUST be capable of extracting a fully qualified domain name from URLs that follow the format defined in [RFC2396]. For the purpose of domain name comparison, the DRM Agent MUST use the mechanism described in Section 1 of [RFC 2965]. If the domain name in the specified URL is in the list of authorised domain names already stored by the DRM Agent for that RI, the DRM Agent MUST attempt to silently acquire the RO for the DCF.
- Before initiating the 4-pass Registration protocol a Device MUST obtain user consent before contacting the RI; however, if the FQDN (Fully Qualified Domain Name) part of the roapURL element of the ROAP Trigger corresponds to an entry in the User Consent Whitelist the Device MAY contact the RI without obtaining explicit user consent.
- Before initiating the 2-pass Identification protocol a Device MUST obtain user consent before contacting the RI; however, if the FQDN (Fully Qualified Domain Name) part of the roapURL element of the ROAP Trigger corresponds to an entry in the User Consent Whitelist the Device MAY contact the RI without obtaining explicit user consent.
- For implied ROAP exchanges as specified in section 5.1.12 a Device MUST obtain user consent in order to contact the RI if it does not have a valid RI Context, however, if the FQDN part of the roapURL element of the ROAP Trigger corresponds to an entry in the User Consent Whitelist the Device MAY contact the RI without obtaining explicit user consent. If a valid RI Context has been established the Device SHOULD NOT obtain explicit user consent for any further implied ROAP exchanges.
- If a Device receives a ROAP response with the status equal to "NotRegistered" or "DeviceTimeError" the Device SHOULD NOT obtain explicit user consent before continuing as specified in section 5.3.6.
- If a Device receives a JoinDomain ROAP Trigger for a Domain that it is not a member of i.e. it does not have a corresponding Domain Context, the Device MAY obtain user consent prior to attempting to join the Domain,
- If a Device receives a JoinDomain ROAP Trigger for a Domain that it is a member of, i.e. it has a corresponding Domain Context, but the Domain Generation is greater than that stored in the corresponding Domain Context, the Device SHOULD NOT obtain explicit user consent prior to attempting to upgrade the Domain.
- If a Device receives a LeaveDomain ROAP Trigger for a Domain that it is not a member of i.e. it does not have a corresponding Domain Context, the Device MUST obtain user consent prior to initiating ROAP, however, if the FQDN part of the roapURL element of the ROAP Trigger corresponds to an entry in the User Consent Whitelist the Device MAY initiate ROAP without obtaining explicit user consent.
- If a Device receives an *Extended Leave Domain* ROAP Trigger (see section 5.2.1.1) that does not contain any **<deviceID>** element, the Device MUST acquire user consent before initiating 2-pass Leave Domain protocol. If the *Extended Leave Domain* trigger does contain a **<deviceID>** element then the Device MUST NOT ask for user consent before initiating the 2-pass Leave Domain protocol,.
- If a Device is attempting to install a Device RO and it determines that it does not have a valid RI Context for the RI as identified by the <riID> element in the roap:ROPayload of a Device RO, the Device MUST obtain user consent prior to contacting the RI, this applies to both ROAP and non-ROAP communications e.g. HTTP GET requests (see section 10.3.1.3), however, if the FQDN part of the roapURL element of the ROAP Trigger or the riURL attribute of the roap:protectedRO corresponds to an entry in the User Consent Whitelist the Device MAY contact the RI without obtaining explicit user consent.
- If a Device is attempting to install a Domain RO and it determines that it is not a member of the Domain for which the Domain RO is issued, i.e. it does not have a corresponding Domain Context, the Device MUST obtain user consent prior to attempting to join the Domain, this applies to both ROAP and non-ROAP communications e.g. HTTP GET requests (see section 8.7.2.1), however, if the FQDN part of the roapURL element of the ROAP Trigger or the riURL attribute of the roap:protectedRO corresponds to an entry in the User Consent Whitelist the Device MAY attempt to join the Domain without obtaining explicit user consent.

- If a Device is attempting to install a Domain RO and it determines that it is a member of the Domain for which the Domain RO is issued, but the Domain Generation is greater than that stored in the corresponding Domain Context, the Device SHOULD NOT obtain explicit user consent prior to attempting to upgrade the Domain, this applies to both ROAP and non-ROAP communications e.g. HTTP GET requests (see section 8.7.2.1).
- The DRM Agent MUST obtain user consent for the collection of Metering Information on a per RI basis:
    - The exception to this rule is when a Device consumes an RO that contains the **<tracked>** element and the riID of the ROAP:ROPayload corresponds to an entry in the User Consent Whitelist, in this case the Device MUST NOT obtain explicit user consent for the collection of Metering Information generated when DRM Content is accessed via the RO. Note: RIs that have entries in the User Consent Whitelist must obtain the appropriate user consent via means that are outside of the scope of this specification e.g. as part of a purchase/subscription.process.
    - If a Device consumes an RO that contains the **<tracked>** element and the riID of the ROAP:ROPayload does not correspond to an entry in the User Consent Whitelist the Device MUST obtain explicit user consent for the collection of Metering Information to the RI identified by the riID. If available the Device MUST use the riAlias when asking the User for consent.
    - If a User has given or denied consent for the collection of Metering Information for a particular RI the Device MUST NOT request user consent every time ROs issued by that RI are consumed. However, users SHOULD be given an opportunity to retract their decision. If user consent was denied and an RO with a <**tracked**> element is consumed and the *contentAccessGranted* attribute is "false" then user consent MAY be re-requested. Otherwise it is recommended that Devices include a configuration option to periodically offer the user the option to revise their decision (for example once a month the metering confirmation for each RI can be revised).
    - If a Device receives a MeteringReport ROAP Trigger and Metering Information has been collected, the Device DMUST NOT obtain explicit user consent prior to sending the Metering Report.
- Before initiating the 2-pass RO Upload protocol the Device MUST obtain user consent. RO upload can be initiated by ROAP Trigger or by user interaction with the Device.

The means for provisioning and management of the User Consent Whitelist implemented for the purpose of determining if explicit user confirmation is required are outside the scope of this specification.

For the purpose of comparing FQDNs and User Consent Whitelist entries, the DRM Agent MUST use the mechanism described in Section 1 of [RFC 2965].

5.2 Initiating the ROAP

5.2.1 The ROAP Trigger

The **ROAPTrigger** type is a sequence of a chosen ROAP trigger (see below), an optional signature on the ROAP trigger, and an optional **<encKey>** element containing a wrapped MAC key. The purpose of a ROAP trigger is to initiate a particular ROAP protocol.

```
<complexType name="BasicRoapTrigger">
  <sequence>
     <element name="riID" type="roap:Identifier"/>
     <element name="riAlias" type="roap:String80" 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"/>
    <element name="domainAlias" type="roap:String80" minOccurs="0"/>
   </sequence>
  </extension>
 </complexContent>
</complexType>

<complexType name="ROAcquisitionTrigger">
 <complexContent>
  <extension base="roap: BasicRoapTrigger">
   <sequence>
    <element name="domainID" type="roap:DomainIdentifier" minOccurs="0"/>
    <element name="domainAlias" type="string" minOccurs="0"/>
    <sequence maxOccurs="unbounded">
     <element name="roID" type="ID"/>
     <element name="roAlias" type="roap:String80" 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="roap:String80" 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:BasicRoapTrigger"/>
   <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>

<simpleType name="String80">
  <restriction base="string">
    <maxLength value="80"/>
  </restriction>
</simpleType>
```

The <**riID**> element identifies the RI as specified in Section 5.4.2.2.1. For triggers besides the <**registrationRequest**> and the <**identificationRequest**>, the DRM Agent MUST use this value to verify that it has a valid RI Context with the Rights Issuer. If the DRM Agent does not have a valid RI Context with the identified Rights Issuer then the DRM Agent MUST initiate the Registration Protocol before initiating the protocol indicated in the <**roapTrigger**> element. If the implicitly triggered Registration Protocol does not lead to a valid RI Context, then the DRM Agent MUST discard the trigger.

The <**riAlias**> element SHOULD contain a String value that SHALL be used by the DRM Agent whenever it refers to the Rights Issuer in a dialog with the user and it SHALL be saved in the RI Context for future use. An example for such a dialog would be the question whether or not the user would like to register with a certain RI after receiving ROAP Trigger. The maximum length of this element SHALL be 80 bytes.

In order to prevent name-spoofing attacks, the DRM Agent SHALL also display a fully qualified domain name of a URL (as defined in [RFC2396]) along with the <**riAlias**>, if present. In case of an existing valid RI Context with the RI identified by the <**riID**> value, the URL whose domain name is to be shown MUST be the RI URL found in the RI Context. If there is no existing valid RI Context, the <**roapURL**> MUST be used instead.

The <**nonce**> element provides a way to couple ROAP triggers with ROAP requests. RIs MUST include a <**nonce**> element in "LeaveDomain" triggers. If the value of the **triggerNonce** attribute in the subsequent Leave Domain Request is not equal to the <**nonce**> element in the "LeaveDomain" trigger, the RI MUST discard the received Leave Domain Request. RIs MUST follow the guidelines for nonces as expressed in Section 5.3.12.

If the ROAP trigger holds a <nonce> element, a Device MUST include the <nonce> value as a triggerNonce attribute in all subsequent ROAP request PDUs.

The DRM Agent MUST use the URL specified by the <**roapURL**> element when initiating the ROAP transaction. The <**roapURL**> is used in conjunction with the protocol indicated in the <**roapTrigger**> element The <**roapURL**> is also used in conjunction with any other implicitly triggered ROAP protocol (See 5.1.12). The bulleted list below describes which ROAP PDU is explicitly selected by the <**roapTrigger**> element.

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 triggered 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 for future ROAP triggers. 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.

If the <**roapTrigger**> element carries a <**registrationRequest**> element, the PDU MUST be a ROAP-DeviceHello PDU.

The XML representation of the Identification Request trigger is defined by re-using the **ExtendedRoapTrigger** type and assigning the fixed value “**identificationRequest**” for the **type** attribute. It SHALL be signalled as an **<extendedTrigger>** element. Consequently, if the **<roapTrigger>** element carries an **<extendedTrigger >** element as defined above with the **type** attribute set to “**identificationRequest**”, the PDU MUST be a ROAP-IdentificationRequest PDU.

If the **<roapTrigger>** element carries an **<roAcquisition>** element, the PDU MUST be a ROAP-RORequest PDU.

If the **<roapTrigger>** element carries a **<joinDomain>** element, the PDU MUST be a ROAP-JoinDomain PDU.

If the **<roapTrigger>** element carries a **<leaveDomain>** element, the PDU MUST be a ROAP-LeaveDomain PDU.

The XML representation of the RO Upload request trigger is defined by re-using the **ExtendedRoapTrigger** type and assigning the fixed value “**roUploadRequest**” for the **type** attribute. It SHALL be signalled as an **<extendedTrigger>** element. Consequently, if the **<roapTrigger>** element carries an **<extendedTrigger>** element as defined above with the **type** attribute set to **”roUploadRequest”**, the PDU MUST be a ROAP-ROUploadRequest PDU.

The XML representation of the Metering Report trigger is defined by re-using the **ExtendedRoapTrigger** type and assigning the fixed value “**meteringReport**” for the **type** attribute. It SHALL be signalled as an **<extendedTrigger>** element. Consequently, if the <roapTrigger> element carries an <**extendedTrigger**> element as defined above with the **type** attribute set to “**meteringReport**”, the PDU MUST be a ROAP-MeteringReportSubmit PDU. If the ROAP trigger holds a <nonce> element, a Device MUST include the <nonce> value as a triggerNonce attribute in the ROAP-MeteringReportSubmit PDU. If the ROAP trigger holds a <nonce> element, and a ROAP protocol is implicitly triggered (See 5.1.12 for protocols), a Device MUST include the <nonce> value as a triggerNonce attribute in the first ROAP request PDU of the implicitly triggered protocol and also the subsequent explicitly triggered ROAP PDU.

The **<domainID>** element MAY be included in certain ROAP triggers. If included, the Device MUST incorporate the **<domainID>** in the ROAP PDU that is sent in response to the trigger. If the **<roapTrigger>** element carries an **<roAcquisition>** element that includes a **<domainID>** element and the Device does not have a valid Domain Context for the Domain indicated by that domain identifier, the Device SHOULD initiate the 2-pass Join Domain Protocol before initiating the RO acquisition protocol. If the **status** attribute of the ROAP-JoinDomainResponse message of the implicitly triggered Join Domain Protocol is not equal to “Success”, then the DRM agent MUST discard the trigger, and when possible, the Device SHOULD present an appropriate error message to the user, using the value of the Status attribute, the **errorMessage** attribute (if present) and/or the **errorRedirectURL** (if present). See section 5.3.5.

Before a Device performs this implicitly triggered JoinDomain ROAP exchange, it may have to obtain user consent, section 5.1.13 defines when explicit user consent is required.

The **<domainAlias>** element SHOULD be included in Triggers that contain a **<DomainID>** element. The **<domainAlias>** element contains a String value that SHALL be used by the DRM Agent whenever it refers to the Domain specified by **<domainID>** in a message to the user. The content of the **<domainAlias>** element SHALL be saved in the Domain Context. The maximum length of this element SHALL be 80 bytes.

One or several **<roID>** elements MUST be included in the **<roAcquisition>** trigger to identify the ROs to be acquired. The RI MAY specify more than one **<roID>** element to initiate download of multiple ROs. The DRM Agent MUST include all received <**roID**> elements in the **<roInfo>** portion of the subsequent ROAP-RORequest PDU.

The **<roAlias>** element SHOULD be included in **<roAcquisition>** triggers if the **<roID>** refers to a group, parent or multi-asset RO. The **<roAlias>** contains a String value that SHALL be used by the DRM Agent whenever it refers to the RO specified by **<roID>** in a message to the user. The alias could for example be used when the DRM Agent displays the usage permissions granted by an RO to the user. The **<roAlias>** enables that DRM Agent to avoid displaying complex REL semantics to the user; instead the DRM Agent can display a simple alias like “July Subscription”. The maximum length of this element SHALL be 80 bytes.

A **<contentID>** element MAY be included in the **<roAcquisition>** trigger for each DCF explicitly referenced by the RO, i.e. not for group or parent elements of the RO. In the latter case, no <contentID> element needs to be included as specific DCFs are not referenced and therefore DCF hashes and transaction IDs cannot be used. In the case where a single RO applies to several specific DCFs the <roAcquisition> trigger MUST include a <contentID> element for each DCF. In the case where an

RO applies to one or more Content Objects (in a Multipart DCF or Multi-track PDCF), the trigger SHALL include the <contentID> element of the first Content Object (or Track) in the Multipart DCF (or PDCF). The **<contentID>** elements MUST contain the ContentID as specified in the ContentID field in the Common Header of the Content Object inside the associated (P)DCF [DRMDCF-v2.1].

In case a **<leaveDomain>** element is present, the RI MUST include a **<signature>** element and, with one exception (see below), Devices MUST verify this signature. If the Device cannot verify the signature, the Device SHOULD inform the user and MUST discard the ROAP Trigger.

The only exception to the verification requirement is when the Device is not a member of the identified Domain, and the trigger has been integrity protected with a MAC based on the Domain Key. In this case the Device may have to obtain user consent before initiating ROAP, section 5.1.13 defines when explicit user consent is required. A Device is part of the identified Domain if it has a Domain Context for that Domain and has access to the Domain Key of the identified Domain Generation.

The **<ds:Reference>** element of the **<ds:SignedInfo>** child element of the **<signature>** shall reference the **<leaveDomain>** element by using the same value for the **URI** attribute as the value for the **<leaveDomain>** element's **id** attribute. The **<ds:KeyInfo>** child element of the **<signature>** element shall use its **URI** attribute of the **<ds:RetrievalMethod>** element to reference a wrapped MAC key in the **<encKey>** element, and the signature algorithm (expressed in the **Algorithm** attribute of the **<ds:SignatureMethod>** element) MUST be "http://www.w3.org/2000/09/xmldsig#hmac-sha1". In compliance to the rules of canonicalisation specified in Section 5.3.3, the **<ds:Reference>** element MUST contain a **<ds:Transforms>** element, that contains a single **<ds:Transform>** element that signals the use of the exclusive canonicalisation algorithm without comments.

The **<encKey>** element shall in the case of a "LeaveDomain" trigger be present and shall contain a MAC key wrapped with the current Domain key. The value of the **Id** attribute of this element shall equal the value of the **URI** attribute of the **<ds:RetrievalMethod>** child element of the **<signature>** element as specified above.

The **version** attribute is a <major.minor> representation of the ROAP trigger. For this version of the specification, **version** SHALL be set to "**1.1**". Minor version upgrades must always be backwards compatible.

If present, the **proxy** attribute indicates that the ROAP Trigger is not for the Connected Device but is intended for an Unconnected Device. Upon receipt of a ROAP Trigger containing the **proxy** attribute with the value set to "**true**" a Connected Device that supports the functionality to provide connectivity for Unconnected Devices (as specified in section 18) MUST start the procedures specified in section 14.6.4. If the **proxy** attribute is present but the value is set to "**false**" then Connected Devices MUST treat the ROAP Trigger as if it did not contain the **proxy** attribute.

The MIME type for the ROAP Trigger is "application/vnd.oma.drm.roap-trigger+xml". The file extension for the ROAP trigger is ".ort" for the ROAP Trigger when the ROAP Trigger is stored on the Device as a file. The file extension for the ROAP PDU is ".oru" for the PDU when the ROAP PDU is stored on the Device as a file.

A binary/compact form of the ROAP Trigger is defined using WBXML encoding in section 16.

5.2.1.1 Extended Leave Domain Trigger

An *Extended Leave Domain* trigger enables the initiation of the ROAP 2-pass Leave Domain protocol. The *Extended Leave Domain* trigger enables additional security compared to the basic <leaveDomain> trigger type; with the inclusion of a **<deviceID>** element.

The XML representation of the *Extended Leave Domain* trigger is an **<extendedTrigger>** element of type **roap:ExtendedRoapTrigger** and of which the type-attribute SHALL be assigned the fixed value **"leaveDomain"**. A **<trgLeaveDomain>** element MUST be included in the **<extendedTrigger>** element after the **<roapURL>** element.

```
<element name="trgLeaveDomain">
 <complexType>
  <sequence>
   <element name="deviceID" type="roap:Identifier" minOccurs="0"/>
   <element name="domainAlias" type="roap:String80" minOccurs="0"/>
   <element name="domainID" type="roap:DomainIdentifier"/>
   <any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
 </complexType>
</element>
```

The elements in the **<trgLeaveDomain>** element have the following meaning:

- The <**deviceID**> element MAY be present in the <**trgLeaveDomain**> element. If the <**deviceID**> is present the DRM Agent MUST verify whether the value of the <**deviceID**> matches the hash of the Device's public key info, as it appears in one of the Device’s certificates (i.e. the hash of the complete DER-encoded `subjectPublicKeyInfo` component). If the <**deviceID**> does not match the hash of the Device’s public key info then the DRM Agent MUST discard the trigger. The presence of the <**deviceID**> element affects the User Consent requirements as described in section 5.1.13.

- The <**domainAlias**> and <**domainId**> elements are as per section 5.2.1.

- Devices MUST ignore any unknown elements following the <**domainID**> element.

In the case of an *Extended Leave Domain* the elements within the **<extendedTrigger>** have the following meaning and restrictions:

- The **<riID>** and **<riAlias>** elements are as per section 5.2.1.

- The **<nonce>** is as per section 5.2.1. RIs MUST include a **<nonce>** element in *Extended Leave Domain* triggers.

- The **<roapURL>** element is as per section 5.2.1. When initiating the ROAP transaction the message sent to the **<roapURL>** MUST be a ROAP-LeaveDomain PDU.

In the case of an *Extended Leave Domain* the elements within the **<roapTrigger>** element have the following meaning and restrictions:

- The <**signature**> and <**keyInfo**> elements MUST be present in the <**roapTrigger**> element. The processing of these elements is as described for the <leaveDomain> in section 5.2.1.

5.2.2 Initiating ROAP from a (P)DCF

This section applies ONLY to Connected Devices.

(P)DCFs may be delivered to a Device without any valid rights already existing on the terminal. Additionally rights already issued for a P(DCF) may expire. There are numerous methods defined within the OMA DRM 2.x specification for a DRM Agent to initiate communication with a Rights Issuer to obtain valid rights for content. This section specifies precise processing rules that MUST be implemented all Devices. These processing rules ensure consistent user experience across all Devices.

When DRM content is received on a Device the DRM Agent SHOULD check if there are already valid rights installed for the Content. If there are no valid rights available for the content the DRM Agent SHOULD attempt to obtain valid rights according to the following sequence:

1. Try to install any Rights Objects from the P(DCF) Rights Object Box (see [DRMCF-v2]). During the installation of an embedded rights objects the DRM Agent MAY register to a Rights Issuer (Device ROs) or join a Domain (Domain ROs). The RI registration or Domain join may be implicit or initiated by sending a HTTP GET to the **riURL** attribute of the **roap:ROPayload** as specified in section 8.7.2.1 and 10.3.1.2.

2. If the (P)DCF contains a Silent Header with method of “in-advance” then send a HTTP GET request to the `silent-rights-url` unless a previous request to the URL has returned a HTTP 404 error code (as specified in [DRMCF-v2]) .

When the Device initiates consumption of DRM Content the DRM Agent MUST check if there are already valid rights installed for the Content. If there are no valid rights available for the content the DRM Agent MUST attempt to obtain valid rights according to the following sequence (abort the sequence if valid rights are obtained):

1. Try to install any Rights Objects from the P(DCF) Rights Object Box (see [DRMCF-v2]). During the installation of an embedded rights objects the DRM Agent MAY register to a Rights Issuer (Device ROs) or join a Domain (Domain ROs). The RI registration or Domain join may be implicit or initiated by sending a HTTP GET to the **riURL** attribute of the **roap:ROPayload** as specified in section 8.7.2.1 and 10.3.1.2.

2. If the (P)DCF contains a Silent Header with method of “in-advance” then send a HTTP GET request to the `silent-rights-url` unless a previous request to the URL has returned a HTTP 404 error code (as specified in [DRMCF-v2]).

3. If the (P)DCF contains a Silent Header with method of “on-demand” or a Preview Header with method of “preview-rights” then send a HTTP GET request to either the `preview-rights-url` or the `silent-rights-url` unless a previous request to the URL has returned a HTTP 404 error code (as specified in [DRMCF-v2]). If both headers exist in the DCF then the DRM Agent MUST give priority to the header that appears first in the (P)DCF. If a request to the first header does not provide valid rights then the DRM Agent SHOULD make a request to the second header.

4. If the (P)DCF contains a Preview Header with method of “instant” the DRM Agent MAY allow the user to use the preview element without any rights. NOTE that the preview element MAY be used at any time.

5. If the (P)DCF contains a *RightsIssuerURL* then the DRM Agent MUST send a HTTP GET request to the URL and MUST be prepared to receive either an (X)HTML page, or ROAP Trigger from the RI

At each of the above steps the DRM Agent MUST obtain user consent as defined in section 5.1.13. If the user denies user consent for one operation the DRM Agent SHOULD proceed to the next step. If user consent is required for multiple of the above options the DRM Agent MAY combine the user consent requests into a single request to the user to select their preferred option. For example the DRM Agent may combine the user consent requirements into a single request as to whether the user wants to try to join a Domain to install an available Domain RO or to visit the RightsIssuerURL to obtain completely new rights.

5.2.2.1 (P)DCF Silent and Preview Header

The Silent and Preview Headers are intended to allow Content and Rights Issuers to easily provide rights to target Devices after download or superdistribution. These URLs can be used effectively to enable subscription scenarios and a wide variety of other use cases.

Upon reception of a HTTP GET request to a a (P)DCF `preview-rights-url` or `silent-rights-url` the RI MUST respond with either:

- An HTTP response with the status equal 200 OK and with the HTTP body containing either:
  - a ROAP-Trigger;
  - a Download Descriptor that points to a ROAP Trigger; or
  - a multipart response containing a bundled Download Descriptor and ROAP Trigger.
- Or a HTTP response with status not equal to 200 OK:
  - Rights Issuers MAY stop Devices from making future requests to the `preview-rights-url` or `silent-rights-url` by responding to DRM Agent requests with error code `404 Not Found` [RFC2616].

Rights Issuers MUST only issue a Download Descriptor to a Device that indicates for support [DLOTA] in either UAProf or Accept Headers.

Additionally Rights Issuers MAY stop Devices from making further requests to these URLs by first issuing a ROAP-ROAcqusition trigger and responding to the subsequent roap:RORequest with a *RightsExpired* ROAP status code

On any occasion where the DRM Agent successfully retrieves and installs an RO acquired as a result of a Silent header or Preview header (with method preview-rights) in a (P)DCF, the DRM Agent MUST add the domain name of the silent or preview URL to the list of authorised domain names for that RI, if the domain name is not already present. As specified in section 5.4.2.4.1, a DRM Agent must be capable of storing a minimum of 5 domain names for each RI Context. In the case where a new domain name is to be added to the list and the list of domain names is full, then the last domain name SHOULD be deleted. Each remaining domain name at position n, SHOULD be moved to position n+1 and the new domain name SHOULD be stored in the first position.

5.2.2.2 (P)DCF Rights Issuer URL

The (P)DCF *RightsIssuerURL* is intended to be an online location where a user can purchase or otherwise obtain rights for the P(DCF). Typically the *RightsIssuerURL* points to an online WEB or WAP portal that enables a user to download previously purchased rights or initiate purchase transactions for new rights.

Requests to the *RightsIssuerURL* MUST return either:

- An (X)HTML page;
- A ROAP Trigger;
- A Download Descriptor that points to a ROAP Trigger; or
- A multipart response containing a bundled Download Descriptor and ROAP Trigger.

Rights Issuers MUST only issue a Download Descriptor to a Device that indicates for support [DLOTA] in either UAProf or Accept Headers.

5.3 ROAP XML Schema Basics

5.3.1 Introduction

Core parts of the XML schema for ROAP, found in Appendix A, are explained in this section. Specific protocol message elements are defined in the Section 5.4. Examples are found in Appendix H.1.

The XML format for ROAP messages have been designed to be extensible. However, it is possible that the use of extensions will harm interoperability and therefore any use of extensions should be carefully considered.

XML Types defined in this sub-section are not ROAP messages; rather they provide building blocks that are used by ROAP messages.

5.3.2 General XML Schema Requirements

Some ROAP exchanges rely on the parties being able to compare received values with stored values. Unless otherwise noted, all elements in this document that have the XML Schema "string" type, or a type derived from it, MUST be compared using an exact binary comparison. In particular, ROAP implementations MUST NOT depend on case-insensitive string comparisons, normalisation or trimming of white space, or conversion of locale-specific formats such as numbers.

The ROAP specification does not define a collation or sorting order for attributes or element values. ROAP implementations MUST NOT depend on specific sorting orders for values.

Values of type **datetime** MUST conform to a single lexical representation defined in section 3.2.7 of [XML-Schema]. This lexical representation is the extended format CCYY-MM-DDThh:mm:ssZ where CC denotes the century, YY denotes the year, MM denotes the month, DD denotes the day, T is the date/time separator, hh, mm, ss represent the hour, minute, and second respectively, and Z is the mandatory UTC indicator. For example, 2002-12-31T23:59:59Z represents December 31st, 2002, 23:59:59 UTC.

Devices MUST support at least 256 byte long values for attributes or elements of type **anyURI** in the schemas specified in this specification. Rights Issuers are RECOMMENDED to use values that are less than 256 bytes in length for such elements or attributes.

5.3.3 Canonicalisation & Digital Signatures

This specification makes use of digital signatures and message authentication codes (MACs) to ensure integrity and authenticity of exchanged information. DRM Agents and RIs MUST support RSA-PSS [PKCS-1] as default digital signature scheme but MAY agree to use a different one (see 5.4.2.1). DRM Agents and RIs MUST send all ROAP messages and triggers in canonicalised form. After canonicalisation, DRM Agents and RIs MUST NOT employ any subsequent transformations or modifications to a ROAP message.

Note that all ROAP messages and triggers are XML 1.0 data. ROAP messages and triggers MUST validate against the ROAP schema **Error! Reference source not found.** and MUST NOT use namespace prefixes other than those used in that ROAP schema.

All canonicalisation steps required by this specification MUST be Exclusive Canonicalisation without comments, as specified in [XC14N]. The InclusiveNamespaces PrefixList of this algorithm MUST be empty. This also applies to any canonicalisation step required by any of the specifications that are normatively referred to by this specification, unless such a referred specification explicitly requires a different canonicalisation algorithm.

In case canonicalisation is to be performed on an XML document as a whole or part of a XML document, the effect SHALL be functionally equivalent to the process of parsing the XML document into an XPath node set, applying XPath expression evaluation to select the proper nodes from this node-set, and subsequently applying Exclusive Canonicalisation without comments to produce the octet-string that is subject to further processing.

Note that this specification does not require any implementation to explicitly implement XPath processing, an implementation MAY utilise the fact that received ROAP PDUs are in Exclusive Canonical Form to implement functional equivalences of XPath based processing.

Where applicable in the ROAP messages and triggers, the use of Exclusive Canonicalisation without comments SHALL be signalled explicitly.

5.3.4 The Request type

All ROAP requests are defined as extensions to the abstract Request type. The elements of the **Request** type therefore apply to all ROAP requests. A ROAP requests MAY contain a **triggerNonce** attribute. See further Section 5.2.1.

```
<complexType name="Request" abstract="true">
  <attribute name="triggerNonce" type="roap:Nonce"/>
</complexType>
```

5.3.5 The Response type

All ROAP responses are defined as extensions to the abstract **Response** type. The elements of the **Response** type therefore apply to all ROAP responses. All responses contain a **status** attribute that indicates whether the preceding request was successful or not.

```
<complexType name="Response" abstract="true">
   <attribute name="status" type="roap:String80" use="required"/>
```

```
  <attribute name=”errorMessage” type=”roap:String1024”/>
  <attribute name=”errorRedirectURL” type=”anyURI”/>
</complexType >

<simpleType name="String1024">
 <restriction base="string">
  <maxLength value="1024"/>
 </restriction>
</simpleType>
```

The generation and delivery of any kind of **Response** message is conditioned by the RI and its policies. The RI MAY deny access to its resources. If this happens, the RI MUST close the protocol gracefully by sending the Device the corresponding **Response** message by including an appropriate error code (e.g. AccessDenied). Implementation details of policies by a given RI are out of the scope of this specification.

In case the **status** attribute is not equal to “Success”, the RI MAY add an additional **errorMessage** attribute containing a RI defined description of the error. The maximum length of this attribute SHALL be 1024 bytes. Also, in case the **status** attribute is not equal to “Success”, the RI MAY add an **errorRedirectURL** attribute that points to a support web site enabling the User to recover from the error. When the RI adds an **errorRedirectURL** attribute it MUST also add an **errorMessage** attribute. A Device SHOULD use the value of the **errorMessage** attribute as part of the error message presented to the User. A Device SHOULD also either include the value of the **errorRedirectURL** as part of the error message to the user, or provide the User with an option to be redirected to the **errorRedirectURL** using a browser.

The OMA DRM 2.1 XML schema allows for any string in the **status** attribute, to enable future extention. The currently specified status messages are defined in the **Status** type.

5.3.6 The Status type

The **Status** simple type enumerates all possible error messages.

```
<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"/>
                            <enumeration value=”TriggerExpiredOrInvalid”/>
                            <enumeration value="UnableToDecryptMeteringReport"/>
                            <enumeration value="UnableToValidateMeteringReportMAC"/>
                            <enumeration value="MalformedMeteringReport"/>
                           <enumeration value="UnknownUploadedRO"/>
                            <enumeration value="InvalidUploadedRO"/>
```

```
        </restriction>
</simpleType>
```

Upon transmission or receipt of a message for which Status is not "Success", the default error handling, 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.

When possible, the Device SHOULD present an appropriate error message to the user, using the value of the Status attribute, the **errorMessage** attribute (if present) and/or the **errorRedirectURL** (if present). See section 5.3.5.

These error messages are valid in all ROAP-Response messages unless explicitly stated otherwise.

Abort indicates that the RI rejected the Device’s request for unspecified reasons.

NotSupported indicates that the Device made a request for a feature currently not supported by the RI.

AccessDenied indicates that the Device is not authorised to contact this RI.

NotFound indicates that the requested object was not found. This error is only valid in the ROAP-ROResponse message.

MalformedRequest indicates that the RI failed to parse the Device's request.

UnknownCriticalExtension indicates that a critical ROAP extension used by the Device was not supported or recognised by the RI.

UnsupportedVersion indicates that the Device used a ROAP protocol version not supported by the RI. This error is only valid in the ROAP-RIHello message and ROAP-IdentificationResponse message.

UnsupportedAlgorithm indicates that the Device suggested algorithms that are not supported by the RI (this error should not occur as long as all Devices and all RIs implement the mandatory algorithms, since any negotiation will successfully fall back on these). This error is only valid in the ROAP-RIHello message.

NoCertificateChain indicates that the RI could not verify the signature on a Device request due to not having access to the Device's certificate chain. This error is only valid in the following messages: ROAP-RegistrationResponse, ROAP-ROResponse, ROAP-JoinDomainResponse, ROAP-LeaveDomainResponse, ROAP-MeteringReportResponse and ROAP-ROUploadResponse.

InvalidCertificateChain indicates that the RI could not verify the signature on a Device request due to the certificate chain being invalid in some way (other than the absence of a trusted root certificate which could be used to verify the chain). This error is only valid in the following messages: ROAP-RegistrationResponse, ROAP-ROResponse, ROAP-JoinDomainResponse, ROAP-LeaveDomainResponse, ROAP-MeteringReportResponse and ROAP-ROUploadResponse.

TrustedRootCertificateNotPresent indicates that the RI could not verify the signature on a Device request due to the absence of a trusted root certificate which could be used to verify the chain. This error is only valid in the following messages: ROAP-RegistrationResponse, ROAP-ROResponse, ROAP-JoinDomainResponse, ROAP-LeaveDomainResponse, ROAP-MeteringReportResponse and ROAP-ROUploadResponse.

SignatureError indicates that the RI could not verify the Device's signature. This error is only valid in the following messages: ROAP-RegistrationResponse, ROAP-ROResponse, ROAP-JoinDomainResponse, ROAP-LeaveDomainResponse, ROAP-MeteringReportResponse and ROAP-ROUploadResponse.

DeviceTimeError indicates that a Device request was invalid due to the Device’s DRM Time being inaccurate as assessed by the Rights Issuer. This error is only valid in the following messages: ROAP-ROResponse, ROAP-JoinDomainResponse, ROAP-LeaveDomainResponse, ROAP-MeteringReportResponse and ROAP-ROUploadResponse. The Device SHOULD NOT perform the default error handling. Instead, the Device SHOULD initiate the 4-pass Registration protocol, using the same ROAP URL as from the ROAP Request that resulted in the error response. See also section 5.1.12. Upon successful completion of the 4-pass Registration protocol the Device SHOULD create and send a new instance of the original request (ROAP-RORequest, ROAP-JoinDomainRequest, ROAP-LeaveDomainRequest, ROAP-MeteringReponse or ROAP-ROUploadResponse) including the Device’s updated DRM Time, with all other parameters remaining the same as for the original request. If the **Response** message received after the resend of the original request contains a **status** attribute equal to “*DeviceTimeError*” or “*NotRegistered*”, the Device MUST handle this repeated error using the default error handling and MUST NOT again start a 4-pass Registration. If the Device is unable to successfully re-register with the RI then it SHOULD NOT resend the original request. The Device may have to obtain user consent to contact the RI, section 5.1.13 defines when explicit user consent is required.

NotRegistered indicates that the Device tried to contact an RI which does not have any registration information stored for the Device. This error is only valid in the following messages: ROAP-IdentificationResponse, ROAP-RIHello, ROAP-ROResponse, ROAP-ROConfirmResponse, ROAP-JoinDomainResponse, ROAP-LeaveDomainResponse , ROAP-MeteringReportResponse and ROAP-ROUploadResponse.

If as part of responding to an Identification Trigger a Device receives an IdentificationResponse message with the ***status*** attribute equal to “*NotRegistered”* the Device SHOULD NOT initiate a 4-pass Registration protocol, but SHOULD perform the default error handling. In case of a ROAP-LeaveDomainResponse with the **status** attribute equal to *"NotRegistered"*, the Device SHOULD NOT initiate the 4-pass Registration protocol, but SHOULD perform the default error handling.

For all other cases, when the Device receives a message with the **status** attribute equal to *"NotRegistered"*, itSHOULD initiate the 4-pass Registration protocol, using the same ROAP URL as from the ROAP Request that resulted in the error response. See also section 5.1.12. Upon successful completion of the 4-pass Registration protocol the Device SHOULD create and send a new instance of the original request (ROAP-RORequest, ROAP-JoinDomainRequest, ROAP-LeaveDomainRequest , MeteringReportResponse or ROAP-ROUploadResponse) including the Device’s updated DRM Time, with all other parameters remaining the same as for the original request. If the **Response** message received after the resend of the original request contains a **status** attribute equal to “*DeviceTimeError*” or “*NotRegistered*”, the Device MUST handle this repeated error using the default error handling and MUST NOT again start a 4-pass Registration.The Device may have to obtain user consent to contact the RI, section 5.1.13 defines when explicit user consent is required.

InvalidDCFHash is sent when the RI detects an incorrect DCF hash value in a ROAP-RORequest message. This error is only valid in the ROAP- ROResponse message.

InvalidDomain indicates that the request was invalid due to an unrecognised Domain Identifier. This error is only valid in the following messages: ROAP-ROResponse, ROAP-JoinDomainResponse, and ROAP-LeaveDomainResponse.

DomainFull indicates that no more Devices are allowed to join the Domain. This error is only valid in the ROAP-JoinDomainResponse message.

DomainAccessDenied indicates that the Rights Issuer does not allow the Device access to the Domain, or the Device identifier can not be authorised without more information. This error is only valid in the ROAP-JoinDomainResponse message.

RightsExpired indicates that the requested rights are no longer available (for this Device). It is only valid in a ROAP-RO-Response message. This response code indicates to the Device that it SHOULD NOT make further attempts to acquire these rights. If the session was initiated by a HTTP GET to a DCF Preview rights URL or a DCF Silent rights URL the DRM Agent SHOULD NOT attempt further requests to the initiating URL (for the current media object). The results of any implicit ROAP transactions MUST remain in effect. The Device MAY ignore this status code.

TriggerExpiredOrInvalid indicates that the ROAP Trigger, from which a ROAP session is initiated by the Device, is Expired or Invalid. When a ROAP session is initiated by a Device using a (stored) ROAP Trigger, RI may check if the ROAP Trigger is expired or modified. When a Device gets this Status code, the Device SHOULD delete the stored ROAP Trigger file on the Device. The Device may also show an error message to the user and indicate that the user needs to get the ROAP Trigger again.

UnableToDecryptMeteringReport indicates that the RI was unable to decrypt the Metering Report. This error is only valid in the ROAP-MeteringReportResponse message.

UnableToValidateMeteringReportMAC indicates that the RI was unable to validate the MAC on the Metering Report. This error is only valid in the ROAP-MeteringReportResponse message.

MalformedMeteringReport indicates that the Metering Report was not formatted properly. This error is only valid in the ROAP-MeteringReportResponse message.

UnknownUploadedRO indicates that at least one RO being uploaded was not issued by this RI. This error is only valid in the ROAP-ROUploadResponse message.

InvalidUploadedRO indicates that at least one RO being uploaded is not eligible for backup. For example the RO may be an expired RO, or a Domain RO. This error is only valid in the ROAP-ROUploadResponse message.

5.3.7 The Extensions type

The **Extensions** type is a list of type-value pairs that define optional ROAP features supported by a Device or an RI. Extensions may be sent with any ROAP message. Please see Section 5.4 in this document for applicable extensions. Unless an extension is marked as critical, a receiving party need not be able to interpret it, and a receiving party is always free to disregard any (non-critical) extensions.

```
<complexType name="Extensions">
  <sequence maxOccurs="unbounded">
    <element name="extension" type="roap:Extension"/>
  </sequence>
</complexType>

<complexType name="Extension" abstract="true">
  <attribute name="critical" type="boolean"/>
</complexType>
```

5.3.8 The ExtensionContainer type

The **ExtensionContainer** type inherits from the **Extension** type and 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>
```

5.3.9 The Protected Rights Object type

The **ProtectedRO** type is a sequence of an <**ro**> element of type **roap:ROPayload** and a <**mac**> element carrying a MAC value over the <**ro**> element. ProtectedRO can be sent separately or included in a ROAP-ROResponse message or a DCF.

```
<element name="protectedRO" type="roap:ProtectedRO" form="qualified"/>

<complexType name="ProtectedRO">
  <sequence>
    <element name="ro" type="roap:ROPayload" form="qualified"/>
    <element name="mac" type="ds:SignatureType"/>
  </sequence>
</complexType>
```

The <**ro**> element is described in the next section.

The <**mac**> element provides integrity of the <**ro**> element and key confirmation and is of type **ds:SignatureType** from [XML-DSIG]. In compliance to the rules of canonicalisation specified in Section 5.3.3, the <**ds:Reference**> element MUST contain a <**ds:Transforms**> element, that contains a single <**ds:Transform**> element that signals the use of the exclusive canonicalisation algorithm without comments.

The **URI** attribute of the <**ds:Reference**> element of the <**ds:SignedInfo**> child element of the <**mac**> SHALL reference the <**ro**> element by having the same value as the **id** attribute of the <**ro**> element. The **URI** attribute of the <**ds:RetrievalMethod**> element of the <**ds:KeyInfo**> child element of the <**mac**> SHALL reference a wrapped MAC key in the <**ro**> element's <**encKey**> child element by having the same value as the **Id** attribute of the <**encKey**> element.

Both the <**protectedRO**> element and its child <**ro**> element in a ROAP message MUST appear in qualified form, so it MUST be prefixed with the roap namespace prefix.

The MIME type for the Protected Rights Object is `application/vnd.oma.drm.ro+xml.`

The file extension for the Protected Rights Object MUST be "`.oro`".

5.3.10 The Rights Object Payload type

Values of the **ROPayload** type carries (protected) rights and wrapped keys that can be used to decrypt encrypted portions of the rights.

```
<!-- 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"/>
    <element ref="roap:roPayloadAliases" minOccurs="0"/>
    <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>
```

The <**riID**> element is of type **roap:Identifier** and SHALL identify the issuing RI.

The **<rights>** element is of type **o-ex:rightsType** and MUST be conformant with [DRMREL-v2.1]. The **o-ex:id** attribute of this type SHALL be present.

The **<signature>** element is of type **ds:SignatureType** from [XML-DSIG] and MUST be present when the RO is a Domain RO. If the **<signature>** is included in a Device RO it enables distribution of Device ROs without the use of the RO Acquisition protocol. The **<signature>** element SHOULD NOT be present when the RO is a Device RO and it is delivered using the RO Acquisition protocol. The **URI** attribute of a **<ds:Reference>** element of the **<ds:SignedInfo>** child element of the **<signature>** SHALL reference the **<rights>** element by having the same value as the **o-ex:id** attribute of the **<rights>** element (i.e., when present, the signature SHALL be made at least over the **<rights>** element). In compliance to the rules of canonicalisation specified in Section 5.3.3, the **<ds:Reference>** element MUST contain a **<ds:Transforms>** element, that contains a single **<ds:Transform>** element that signals the use of the exclusive canonicalisation algorithm without comments. The **<ds:KeyInfo>** child element of the **<signature>** element SHALL identify the signing key. The Device MUST verify that the signing key is associated with the RI identified in the **<riID>** element.

The **<timeStamp>** value MUST be given in Universal Coordinated Time (UTC). The time-stamp provides replay protection, see further in section 10.4. RIs MUST include a timeStamp for all Device ROs.

The **<encKey>** element is of type **xenc:EncryptedKeyType** from [XML-Enc]. It consists of a wrapped concatenation of a MAC key, K_{MAC} and an RO encryption key, K_{REK}. If the <rights> element does not contain a <ds:KeyInfo> element (for example if the <rights> element is used as parent right; see REL, section 5.2.2), the RO encryption key, K_{REK}, is still required in the <encKey> element but, it is not used. 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 the **<ds:KeyInfo>** elements (if present) inside the **<rights>** element. The **<ds:KeyInfo>** child element of the **<encKey>** element SHALL identify the wrapping key. In the case of a Rights Object intended for a Device, the child of the **<ds:KeyInfo>** element SHALL be the <**roap:X509SPKIHash>** element, identifying a particular DRM Agent's public key through the (SHA-1) hash of the DER-encoded subjectPublicKeyInfo value in its certificate. In the case of a Rights Object intended for a Domain, it will be of the type <**roap:domainID>** element, identifying the correct Domain key.

Note that the encrypted key material consists of **two** keys - a MAC key and a Rights Object Encryption key. A Rights Object Encryption key MUST be present, even if it is unused in cases where the <rights> object does not contain a <ds:KeyInfo> element. For further information on packaging the MAC key and the Rights Object Encryption key, see the Key Management discussion in section 7.

Further elements MAY be included into the ROPayload after the **<encKey>** element. Devices MUST disregard any unknown elements. A < roPayloadAliases> element SHOULD be included in the ROPayload.

The **version** attribute indicates the version of the ROPayload type. For this version of the OMA DRM specification, the value SHALL be “1.1”. Future versions of OMA DRM may define minor upgrades of the ROPayload and define additional elements to follow the **<encKey>** element. However, minor version upgrades must always be backwards compatible. The ROPayload version must not be confused with the OMA DRM version, which is independently set. The reason for having different versions is to enable Domain ROs to be shared between Devices with different OMA DRM protocol versions.

The **id** attribute of the **ROPayload** type identifies the RO and MUST correspond to an **<roID>** value in the previous ROAP-RORequest, if there was one. The **id** attribute is also used as a reference point for the MAC as described in the previous section. This Rights Object identifier MUST uniquely identify a rights object; i.e. any two ROs sharing the same RO-id MUST be equivalent (have identical Exclusive Canonical forms).

The **stateful** attribute, when present and set to “**true**”, indicates that the RO contains stateful rights (i.e. needs replay protection). The **id** attribute MUST be globally unique when this attribute is present and set to true, in order to enable a Device to correctly enforce replay protection (Note: one way for an RI to generate globally unique identifiers is to combine an RI-unique and freshly generated nonce with the hash of the RI's public key). If the **stateful** attribute is not present, or is set to "**false**", then the RI does not regard the RO as stateful.

The **domainRO** attribute, when present and set to "**true**", indicates that the RO is for a Domain. If the **domainRO** attribute is not present, or is set to "**false**", then the RO is for a particular Device.

The **riURL** attribute, if present, SHALL contain a URL that the Device can use to contact the RI. In case of a Domain RO, a HTTP GET on this URL SHOULD return either a JoinDomain ROAP Trigger or a (X)HTML page that starts an interaction with the User which may eventually lead to a JoinDomain ROAP Trigger. In case of Device RO, an HTTP GET request to this URL SHOULD return either a RegistrationRequest ROAP Trigger or a (X)HTML page that starts an interaction with the User which may eventually lead to a RegistrationRequest ROAP Trigger.The value of the **riURL** MUST be a URL according to [RFC2396], and MUST be an absolute identifier.

5.3.11 The <roPayloadAliases> element

An <**roPayloadAliases**> element SHOULD be included in the ROPayload after the <**encKey**> element.

```
<element name="roPayloadAliases">
  <complexType>
    <sequence>
      <element name="roAlias" type="roap:String80" minOccurs="0"/>
      <element name="domainAlias" type="roap:String80" minOccurs="0"/>
      <element name="riAlias" type="roap:String80"/>
      <any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>
```

The <**riAlias**>, <**roAlias**> and <**domainAlias**> elements are as described for ROAP Triggers in section 5.2.1. The <**riAlias**> SHOULD always be specified. If the <**riAlias**> does not exist in the RI Context, it SHALL be saved in the RI Context for future use. The <**roAlias**> MAY be specified for any RO, but it SHOULD be specified for Group, Parent or multi-asset ROs. The <**domainAlias**> SHOULD be specified for Domain ROs. If the <**domainAlias**> does not exist in the Domain Context, it SHALL be saved in the Domain Context for future use.

Future versions of OMA DRM MAY define additional aliases to be located in the <**roPayloadAliases**> element after the <**riAlias**> element (validating against the <**any**/> element). Unknown elements MUST be disregarded

5.3.12 The Nonce type

The **Nonce** type is used to carry arbitrary values in the ROAP protocol messages. A nonce, as the name implies, must be used only once. For each ROAP message that requires a nonce element to be sent, a fresh nonce SHALL be generated randomly each time. Nonce values MUST be at least 14 octets long. Devices MUST at least support nonce values 14 octets long.

```
<simpleType name="Nonce">
  <restriction base="base64Binary">
     <minLength value="14"/>
  </restriction>
</simpleType>
```

5.4 ROAP Messages

In this section, ROAP protocol suite messages, including their parameters, encodings and semantics are defined. The ROAP protocol messages are divided into five categories: Registration, Identification, RO Acquisition, Domain management and Metering.

5.4.1 Notation

In the message parameter tables below, "M" stands for "mandatory presence" and "O" stands for "optional presence".

5.4.2 Registration Protocol

5.4.2.1 Device Hello

The ROAP-DeviceHello message is sent from the Device to the Rights Issuer to initiate the 4-pass Registration protocol. This message expresses Device information and preferences.

5.4.2.1.1 Message description

Parameter	ROAP-DeviceHello
Version	M
Device ID	M
Supported Algorithms	O
Extensions	O

Table 2: Device Hello Message Parameters

Version is a <major.minor> representation of the highest ROAP version number supported by the Device. Devices MUST support all versions prior to the one they suggest. For this version of the protocol, Version SHALL be set to "**1.1**". Minor version upgrades must always be backwards compatible.

Device ID identifies the Device to the RI. The only identifier currently defined is the hash of the Device's public key info, as it appears in the certificate (i.e. the hash of the complete DER-encoded `subjectPublicKeyInfo` component in the Device's certificate). The default hash algorithm is SHA-1. The Device MUST send at least one Device ID. In case a Device holds multiple public keys, the Device MAY select one or more of these public keys and send the corresponding Device IDs. Other identifiers are allowed but interoperability when using them is not guaranteed.

Supported Algorithms identifies the cryptographic algorithms (hash algorithms, MAC algorithms, signature algorithms, key transport algorithms and key wrap algorithms) that are supported by the Device. Algorithms are identified using common URIs. The following algorithms and associated URIs MUST be supported by all Devices and RIs:

Hash algorithms:

SHA-1: http://www.w3.org/2000/09/xmldsig#sha1

MAC algorithms:

HMAC-SHA-1: http://www.w3.org/2000/09/xmldsig#hmac-sha1

Signature algorithms:

RSA-PSS-Default: http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-1#rsa-pss-default

Key transport algorithms:

RSAES-KEM-KDF2-KW-AES128:

http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-1#rsaes-kem-kdf2-kw-aes128

Key wrapping algorithms:

AES-WRAP: http://www.w3.org/2001/04/xmlenc#kw-aes128

Canonicalisation algorithms:

Exclusive Canonicalisation: http://www.w3.org/2001/10/xml-exc-c14n#

SHA-1 is defined in [SHA-1]. HMAC-SHA-1 is defined in [HMAC]. RSA-PSS-Default is RSASSA-PSS with all parameters having default values (see [PKCS-1] Appendix C). AES-WRAP is defined in [AES-WRAP]. RSA-KEM-

KDF2-KW-AES128 is defined in Section 7, Key Management. Exclusive Canonicalisation is defined in [XC14N], its use is further explained in Section 5.3.3 of this document.

Use of other algorithm URIs is optional. Since all Devices and all RIs must support the algorithms above, they need not be sent. Only URIs for algorithms not in this list needs to be sent in a ROAP-DeviceHello message.

Extensions: There are no extensions defined for the ROAP-DeviceHello message.

5.4.2.1.2 Message syntax

The <**deviceHello**> element specifies the ROAP-DeviceHello message, which is the first message sent in the 4-pass ROAP Registration protocol. It has complex type **roap:DeviceHello**, which extends the basic **roap:Request** type.

```
<element name="deviceHello" type="roap:DeviceHello"/>

<complexType name="DeviceHello">
  <annotation>
    <documentation xml:lang="en">
      Message sent from Device to RI to establish an RI Context.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="roap:Request">
      <sequence>
        <element name="version" type="roap:Version"/>
        <element name="deviceID" type="roap:Identifier"
                 maxOccurs="unbounded"/>
        <element name="supportedAlgorithm" type="anyURI"
                 minOccurs="0" maxOccurs="unbounded"/>
        <element name="extensions" type="roap:Extensions"
                 minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

The following schema fragment defines the **Version** type. As noted above, for this version of ROAP, its value shall be "**1.1**".

```
<simpleType name="Version">
  <restriction base="string">
    <pattern value="\d{1,2}\.\d{1,3}"/>
  </restriction>
</simpleType>
```

The following schema fragment defines the **Identifier** type and its alternatives. Any non-standard identifier value must be expressed in well-formed XML.

```
<complexType name="Identifier">
  <choice>
    <element name="keyIdentifier" type="roap:KeyIdentifier"/>
    <any namespace="##other" processContents="strict"/>
  </choice>
</complexType>

<complexType name="KeyIdentifier" abstract="true"/>
```

A key can be defined by use of a hash of the key. The hash shall in this case be made over the DER-encoded `subjectPublicKeyInfo` value from the applicable certificate.

```
<complexType name="X509SPKIHash">
```

```
  <complexContent>
    <extension base="roap:KeyIdentifier">
      <sequence>
        <element name="hash" type="base64Binary"/>
      </sequence>
      <attribute name="algorithm" type="anyURI" default="http://www.w3.org/2000/09/xmldsig#sha1"/>
    </extension>
  </complexContent>
</complexType>

<!-- The corresponding ds:KeyInfo element -->
<element name="X509SPKIHash" type="roap:X509SPKIHash"/>
```

5.4.2.2 RI Hello

The ROAP-RIHello message is the second message of the Registration protocol and is sent from the Rights Issuer to the Device in response to a ROAP-DeviceHello message. The message expresses RI preferences and decisions based on the values supplied by the Device.

5.4.2.2.1 Message description

Parameter	**ROAP-RIHello**	
	Status = “Success”	**Status ≠ “Success”**
Status	M	M
Session ID	M	-
Selected Version	M	-
RI ID	M	-
Selected Algorithms	O	-
RI Nonce	M	-
Trusted Device Authorities	O	-
Server Info	O	-
Extensions	O	-

Table 3: RI Hello Message Parameters

Status indicates if the ROAP-DeviceHello request was successfully (Status = Success) handled or not. In the latter case an error code as specified in Section 5.3.6 is sent.

Session ID is a protocol session identifier set by the RI. This allows for several, concurrent, RI-Device sessions.

Selected Version is the selected ROAP version. The selected version will be min(Device suggested version, highest version supported by RI). This information is part of the RI Context.

RI ID identifies the RI to the Device. The only identifier currently defined is the hash of the Rights Issuer’s public key info, as it appears in the certificate (i.e. the hash of the complete DER-encoded `subjectPublicKeyInfo` component in the Rights Issuer’s certificate). The default hash algorithm is SHA-1. This information is part of the RI Context.

Selected Algorithms specifies the cryptographic algorithms (hash algorithm, signature algorithm, MAC algorithm and key transport algorithm) to use in subsequent ROAP interactions. If the Device indicated support of only mandatory algorithms (i.e. left out the <**supportedAlgorithms**> element), or the RI only supports the mandatory algorithms, then the RI need not send this field. Otherwise, the RI MUST provide this parameter and MUST identify one algorithm of each type. This information is part of the RI context.

RI Nonce is a random nonce sent by the RI. Nonces are generated and used in this message as specified in section 5.3.12.

Trusted Device Authorities is a list of Device trust anchors recognised by the RI. This parameter is optional. The parameter is not sent if the RI already has the Device's certificate or otherwise is able to verify a signature made by the Device. If the parameter is present but empty, it indicates that the Device is free to choose any Device certificate to authenticate itself. Otherwise the Device MUST choose a certificate chaining back to one of the recognised trust anchors. Trust anchors are identified in the same manner as Devices and RIs.

Server Info contains server-specific information that the Device must return unmodified, in the ROAP-RegistrationRequest. The Device must not attempt to interpret the value of this parameter. Devices MUST support the Server Info element being of length 512 bytes and MAY support Server Info elements of length greater than 512 bytes. RIs SHOULD keep Server Info length to 512 bytes or less.

Extensions: The following extensions are defined for the ROAP-RIHello message:

Peer Key Identifier: An identifier for a Device public key stored by the RI. If the identifier matches one of the Device ID's in the preceeding DeviceHello message, it means the RI has already stored that Device ID and the corresponding Device certificate chain, and the Device need not send that certificate chain in a later request message. The RI may choose exactly one of the Device ID's from the preceeding DeviceHello message to be included in the RIHello message. Keys are identified in the same way as Devices are (a hash of the DER-encoded `subjectPublicKeyInfo` component of the Device's certificate). If the RI has stored the Device public key the RI MUST use this extension in the ROAP-RIHello. This extension also informs the Device that the RI has the capability to store information about Device certificates.

Certificate Caching: When present, this extension indicates to the Device that the RI has the capability to store information about the Device certificate and that Device certificate chain sending is not necessary in subsequent protocol instances once the RI has received the Device certificate chain. This extension is not needed if the Peer Key Identifier is used, since the latter contains even more specific information.

Device Details: By including this extension, the RI requests the Device to return Device-specific information such as manufacturer and model in a subsequent ROAP-RegistrationRequest message. When present, the DeviceDetails extension SHALL be empty (i.e. <**extension xsi:type="roap:DeviceDetails"**/>)".

If the RI has capabilities to store Device certificates, then the RI MUST send either the *Peer Key Identifier* or the *Certificate Caching* extension in its ROAP-RIHello message. If the ROAP-RIHello contains a *Peer Key Identifier* extension, it SHOULD NOT contain a *Certificate Caching* extension.

The Device SHOULD note in the RI Context whether the RI has a correct public key for the Device stored and/or whether the RI has the capability to store information about the Device's certificate.

5.4.2.2.2 Message syntax

The <**riHello**> element specifies the ROAP-RIHello message, which is sent in response to the ROAP-DeviceHello message. It has complex type **roap:RIHello.**

```
<element name="riHello" type="roap:RIHello"/>

<complexType name="RIHello">
  <annotation>
    <documentation xml:lang="en">
      Message sent from RI to Device in response to a deviceHello message.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="roap:Response">
      <sequence minOccurs="0">
        <element name="selectedVersion" type="roap:Version"/>
        <element name="riID" type="roap:Identifier"/>
        <element name="selectedAlgorithm" type="anyURI" maxOccurs="unbounded"
            minOccurs="0"/>
        <element name="riNonce" type="roap:Nonce"/>
        <element name="trustedAuthorities" type="roap:KeyIdentifiers" minOccurs="0"/>
        <element name="serverInfo" type="base64Binary" minOccurs="0"/>
        <element name="extensions" type="roap:Extensions" minOccurs="0"/>
      </sequence>
      <attribute name="sessionId" type="roap:String64"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="KeyIdentifiers">
  <sequence minOccurs=”0” maxOccurs="unbounded">
    <element name="keyIdentifier" type="roap:KeyIdentifier"/>
  </sequence>
</complexType>

<simpleType name="String64">
  <restriction base="string">
    <maxLength value="64"/>
  </restriction>
</simpleType>
```

The maximum length of the sessionId attribute SHALL be 64 bytes.

The following schema fragment defines the Peer Key Identifier extension:

```
<complexType name="PeerKeyIdentifier">
  <complexContent>
    <extension base="roap:Extension"/>
       <element name="identifier" type="roap:KeyIdentifier"/>
    </extension>
  </complexContent>
</complexType>
```

The following schema fragment defines the *Certificate Caching* extension:

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

The following schema fragment defines the *Device Details* extension:

```
<complexType name="DeviceDetails">
  <complexContent>
    <extension base="roap:Extension">
      <sequence minOccurs=”0”>
       <element name="manufacturer" type="roap:String64"/>
       <element name="model" type="roap:String64"/>
       <element name="version" type="roap:String64"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

The maximum length of the <manufacturer> SHALL be 64 bytes. The maximum length of the <model> SHALL be 64 bytes. The maximum length of the <version> SHALL be 64 bytes.

5.4.2.3 Registration Request

A Device sends the ROAP-RegistrationRequest message to an RI to request registration with the RI. The message is sent as the third message in the 4-pass Registration protocol.

5.4.2.3.1 Message description

Parameter	ROAP-RegistrationRequest
Session ID	M
Device Nonce	M
Request Time	M
Certificate Chain	O
Trusted RI Authorities	O
Server Info	O
Extensions	O
Signature	M

Table 4: Registration Request Message Parameters

Session ID SHALL be identical to the Session ID parameter of the preceding ROAP-RIHello message, otherwise the RI SHALL terminate the Registration protocol.

Device Nonce is a nonce chosen by the Device. Nonces are generated and used in this message as specified in section 5.3.12.

Request Time is the current DRM Time as measured by the Device. Connected Devices and Unconnected Devices that support DRM Time MUST insert their current DRM Time. If the DRM Agent is not previously registered with the Rights Issuer, then the Rights Issuer’s trust model is unknown; therefore the DRM Agent is free to choose any available time to send as the *RequestTime*. If the DRM Agent has previously registered with the RI the DRM Agent MUST choose the DRM Time which corresponds to the trust model previously used by the RI. Unconnected Devices that do not support DRM Time MUST use the value “**Undefined**”.

Certificate Chain: This parameter MUST be present unless the preceding ROAP-RIHello message contained the *Peer Key Identifier* extension and its value identified the key in the Device's current certificate. When present, the value of a *Certificate Chain* parameter shall be a certificate chain including the Device's certificate. The chain SHALL not include the root certificate. The Device certificate must come first in the list. Each following certificate must directly certify the one preceding it. If the RI indicated trust anchor preferences in the previous ROAP-RIHello message, the Device MUST select a Device certificate and chain which chains back to one of the trust anchors indicated by the RI. This mimics the features of [RFC3546]. The RI may need to update this information based on the received *Certificate Chain*.

Trusted RI Authorities is a list of RI trust anchors recognised by the Device. If the parameter is empty, it indicates that the RI is free to choose any certificate. Trust anchors are identified in the same way as Devices and RIs.

Server Info: As discussed above, this parameter will only be present if a *Server Info* parameter was present in the preceding ROAP-RIHello message. In that case, the *Server Info* parameter MUST be present and MUST be identical to the Server Info parameter received in the preceding ROAP-RIHello message.

Extensions: The following extensions are defined for the ROAP-RegistrationRequest message:

> *Peer Key Identifier:* An identifier for an RI public key stored in the Device. If the identifier matches the RI ID in the preceding RI Hello message, it means the RI need not send down its certificate chain in its response message. Keys are identified in the same way as Devices and RIs.

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 a trusted OCSP responder key stored in the Device and is used to save bandwidth. 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.

Device Details: This extension defines three fields: manufacturer, model and version. The manufacturer field identifies the Device' manufacturer, the model field identifies the Device's model and the version field identifies the Device's version as defined by its manufacturer. This extension MUST be supported and MUST be sent by a Device that receives an empty *Device Details* extension in a ROAP-RIHello message.

The Device MUST send the *Peer Key Identifier* extension if, and only if, it has stored the RI public key corresponding to the RI ID in the preceding RI Hello message. 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 is a signature on data sent so far in the protocol. The signature is made using the Device's private key on the two previous messages (ROAP-DeviceHello, ROAP-RIHello) and the current message (besides the *Signature* element itself). The signature method is as follows:

- The previous messages and the current one except the *Signature* element is canonicalised according to Section 5.3.3.
- The three messages are concatenated in their chronological order, starting with the ROAP-DeviceHello message. 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 RI MUST verify the signature on the ROAP-RegistrationRequest message.

5.4.2.3.2 Message syntax

The <**registrationRequest**> element specifies the ROAP-RegistrationRequest message, which is the third message in the ROAP Registration protocol. It has complex type **roap:RegistrationRequest**, which extends the basic **roap:Request** type.

```
<element name="registrationRequest" type="roap:RegistrationRequest"/>

<complexType name="RegistrationRequest">
  <annotation>
    <documentation xml:lang="en">
      Message sent from Device to RI to request registration.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="roap:Request">
      <sequence>
        <element name="nonce" type="roap:Nonce"/>
        <element name="time" type="roap:dateTimeOrUndefined"/>
        <element name="certificateChain" type="roap:CertificateChain"
                 minOccurs="0"/>
        <element name="trustedAuthorities" type="roap:KeyIdentifiers"
                 minOccurs="0"/>
        <element name="serverInfo" type="base64Binary" minOccurs="0"/>
        <element name="extensions" type="roap:Extensions" minOccurs="0"/>
        <element name="signature" type="base64Binary"/>
      </sequence>
      <attribute name="sessionId" type="roap:String64" use="required"/>
    </extension>
  </complexContent>
```

```
</complexType>
```

The maximum length of the sessionId attribute SHALL be 64 bytes.

The <**time**> element expresses, in UTC, the current DRM Time as measured by the Device. The value "**Undefined**" is used by Unconnected Devices that do not support DRM Time.

```
<simpleType name="dateTimeOrUndefined">
  <union memberTypes="dateTime roap:UndefinedString"/>
</simpleType>

<simpleType name="UndefinedString">
  <restriction base="string">
  <enumeration value="Undefined"/>
  </restriction>
</simpleType>
```

The following schema fragment defines the **CertificateChain** type, containing a sequence of one or more base64-encoded X.509 certificates in DER-encoded form.

```
<complexType name="CertificateChain">
  <sequence maxOccurs="unbounded">
    <element name="certificate" type="base64Binary"/>
  </sequence>
</complexType>
```

The following schema fragment defines the extensions defined for the ROAP-RegistrationRequest message (besides the Peer Key Identifier and Device Details extensions already defined earlier in this document):

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

<complexType name="OCSPResponderKeyIdentifier">
  <complexContent>
    <extension base="roap:Extension">
        <sequence>
          <element name="identifier" type="roap:KeyIdentifier"/>
        </sequence>
    </extension>
  </complexContent>
</complexType>
```

5.4.2.4 Registration Response

The ROAP-RegistrationResponse message is sent from the Rights Issuer to the Device in response to a ROAP-RegistrationRequest message. This message completes the Registration protocol, and if successful, enables the Device to establish an RI Context for this RI.

5.4.2.4.1 Message description

Parameter	**ROAP-RegistrationResponse**	
	Status = "Success"	***Status ≠ "Success"***
Status	M	M
Session ID	M	O
RI URL	M	-

Certificate Chain	O	-
OCSP Response	O	-
Extensions	O	-
Signature	M	-

Table 5: Registration Response Message Parameters

Status indicates if the ROAP-RegistrationRequest message was successfully (Status = Success) handled or not. In the latter case an error code as specified in Section 5.3.6 is sent.

Session ID SHALL be identical to the Session ID of the preceding ROAP-RegistrationRequest (and ROAP-RIHello) message. If the Session ID of the ROAP-RegistrationResponse does not equal the Session ID of the corresponding ROAP-RIHello, the Device MUST terminate the protocol. The Session ID can be present only if the Rights Issuer could detect the session identifier in the registration request.

RI URL: if the ROAP-RegistrationRequest message was successful (Status=Success) then the RI URL parameter indicates the ROAP URL that SHOULD be stored in the RI Context. This URL can be used by the Device in later interactions with the RI to send ROAP requests. Section 5.1.12 defines the rules for ROAP URL selection. The value of the parameter MUST be a URL according to [RFC2396], and MUST be an absolute identifier.

Certificate chain: This parameter MUST be present unless the preceding ROAP-RegistrationRequest 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 a certificate chain including the RI's certificate. The chain MUST NOT include the root certificate. The RI certificate must come first in the list. Each following certificate must directly certify the one preceding it. If the Device indicated trust anchor preferences in its ROAP-RegistrationRequest message, the RI SHOULD select a certificate and chain which chains back to one of the trust anchors in the Device's list. This mimics the features of [RFC3546]. For security reasons the Device MUST discard the Registration Response if the hash of the complete DER-encoded subjectPublicKeyInfo component in the received RI certificate does not match the value of the RI ID from the preceding RI Hello message.

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 MUST uniquely identify the RI certificate and MUST 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 MUST verify the RI certificate chain according to Section 6.2.

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 Connected Devices and Unconnected 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 MUST verify the complete chain according to Section 6.2 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 MUST store the expiry time of the RI's certificate (as indicated by the notAfter field within the certificate) in the RI Context and MUST 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 MUST abandon processing the RI message and MUST initiate the registration protocol.

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). An exception to this is when the RI deems that the Device's DRM Time is inaccurate. For the processing of this parameter, see further in Section 6.

Extensions: The following extensions are defined for the ROAP-RegistrationResponse message.

> *Domain Name Whitelist:* This extension allows an RI to specify a list of fully qualified domain names (as defined in [RFC 2396]) that are to be regarded as trusted for the purposes of Silent and Preview headers. The Device MUST store the domain names in the RI Context for this RI. The Device MUST be able to use these domain names for processing DCFs containing the Silent header or a Preview header with method "preview-rights" and a specified preview URL, as defined in section 5.2.2 of this document. The Device MUST treat each domain name received in the Domain Name Whitelist as if it were a fully qualified domain name that had been extracted from an RI URL according to the conditions defined in section 5.2.2 of this document. The Device MUST be capable of storing a minimum of 5 fully qualified domain names for each RI Context supported on the Device.

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

- The previous message as received (that is, including the *Signature* element) and the current one except the Signature element is canonicalised according to Section 5.3.3.
- The two messages are concatenated in their chronological order, starting with the ROAP-RegistrationRequest message. 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 Registration protocol 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 registration failed the Device MUST NOT store the RI Context for this RI, otherwise the Device SHOULD store the RI Context for this RI.

The stored RI Context SHALL at a minimum contain: Device ID, riURL, RI ID, Selected Version, Selected Algorithms, a Certificate Caching indication if the RI has stored the Device certificate or not , and a reference to the DRM Time for the trust model of the RI. The RI Context MAY also contain RI certificate validation data, OCSP responder key and the current set of OCSP responses. The RI Context SHALL also contain an RI Context Expiry Time, which is defined to be the RI certificate expiry time. If the registration process has started with a Registration Trigger that contained the **<riAlias>** element, the RI Context SHALL also contain the riAlias. For Unconnected Devices that do not support DRM Time, the RI Context is infinite i.e., it does not have an expiry time. If the RI Context has expired, the Device MUST NOT execute any other protocol than the 4-pass Registration protocol with this RI, and upon detection of RI Context expiry the Device SHOULD initiate the Registration protocol using the URL as defined by the selection mechanism in section 5.1.1210. The Device SHALL have at most one RI Context with each RI. An existing RI Context SHALL be replaced with a newly established RI Context after successful re-registration with the same RI.

Note that any cached OCSP responses have their own validity period, which normally will be much shorter than the validity period of the RI Context. Per [OCSP-MP], if an OCSP response does not have the nextUpdate present, then the DRM Agent MUST not cache the OCSP response.

Devices and Rights Issuers MUST store the Device ID and RI ID that have been negotiated after the successful registration protocol run.

5.4.2.4.2 Message syntax

The <**registrationResponse**> element specifies the ROAP-RegistrationResponse message, and constitutes the last message in the Registration protocol. It has complex type **roap:RegistrationResponse**, which extends the basic **roap:Response** type.

```
<element name="registrationResponse" type="roap:RegistrationResponse"/>

<complexType name="RegistrationResponse">
  <annotation>
    <documentation xml:lang="en">
      Message sent from RI to Device in response to a registrationRequest message.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="roap:Response">
      <sequence minOccurs="0">
        <element name="riURL" type="anyURI"/>
        <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>
      <attribute name="sessionId" type="roap:String64"/>
    </extension>
  </complexContent>
</complexType>
```

The maximum length of the sessionId attribute SHALL be 64 bytes.

The following schema fragment defines the Domain Name Whitelist extension:

```
<complexType name="DomainNameWhiteList">
  <complexContent>
    <extension base="roap:Extension">
      <sequence maxOccurs="5">
        <element name="dn" type="roap:String80"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

The maximum length of the <dn> SHALL be 80 bytes.

5.4.3 Identification Protocol

If a Device receives an Identification Trigger the Device MUST respond with an IdentificationRequest message and the RI MUST respond to this message with an IdentificationResponse message. If the RI determines that it does not have any registration information for the Device it SHOULD send an IdentificationResponse message with the ***status*** attribute equal to "*NotRegistered*".

5.4.3.1 Identification Request

The ROAP-Identification message is sent from the Device to the Rights Issuer to initiate the 2-pass Identification protocol. This message contains Device specific information.

5.4.3.1.1 Message description

Parameter	ROAP-IdentificationRequest
Version	M
Device ID	M
Extensions	O

Table 6: Identification Request Message Parameters

Version SHALL be identical to the *Version* parameter as described in Device Hello message.

Device ID SHALL be identical to the *Device ID* parameter as described in Device Hello message.

Extensions: There are no extensions defined for the ROAP-IdentificationRequest message.

5.4.3.1.2 Message syntax

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

```
<element name="identificationRequest" type="roap:IdentificationRequest"/>

<complexType name="IdentificationRequest">
  <annotation>
    <documentation xml:lang="en">
      Message sent from Device to RI to identify the Device.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="roap:Request">
      <sequence>
        <element name="version" type="roap:Version"/>
        <element name="deviceID" type="roap:Identifier"
                 maxOccurs="unbounded"/>
        <element name="extensions" type="roap:Extensions"
                 minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

5.4.3.2 Identification Response

The ROAP-IdentificationResponse message is sent from the Rights Issuer to the Device in response to a ROAP-IdentificationRequest message.

5.4.3.2.1 Message description

Parameter	ROAP-IdentificationResponse
Status	M

Table 7: Identification Response Message Parameters

Status indicates if the ROAP-IdentificationRequest message was successfully (Status = *Success*) handled or not. In the latter case an error code as specified in Section 5.3.6 is sent.

5.4.3.2.2 Message syntax

The <**identificationResponse**> element specifies the ROAP-IdentificationResponse message. It has complex type **roap:IdentificationResponse**.

```
<element name="identificationResponse" type="roap:IdentificationResponse"/>

<complexType name="IdentificationResponse">
  <annotation>
    <documentation xml:lang="en">
      Message sent from RI to Device in response to an IdentificationRequest message.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="roap:Response">
    </extension>
  </complexContent>
</complexType>
```

5.4.4 RO Acquisition

5.4.4.1 RO Request

The ROAP-RORequest message is sent from a Device to an RI to request Rights Objects. This message is the first message of the 2-pass RO Acquisition protocol.

5.4.4.1.1 Message description

ROAP-RORequest	
Parameter	**Mandatory/Optional**
Device ID	M
Domain ID	O
RI ID	M
Device Nonce	M
Request Time	M
RO Info	M
Certificate Chain	O
Extensions	O
Signature	M

Table 8: RO Request Message Parameters

Device ID identifies the requesting Device. The value MUST equal the stored Device ID as specified in Section 5.4.2.4.1.

Domain ID, when present, identifies the Domain for which the requested ROs shall be issued.

RI ID identifies the authorizing RI. The value MUST equal the stored RI ID as specified in Section 5.4.2.4.1.

Device Nonce is a nonce chosen by the Device. Nonces are generated and used in this message as specified in section 5.3.12.

Request Time is the current DRM Time, as seen by the Device.

RO Info identifies the requested Rights Object(s). The parameter consists of a (non-empty) set of Rights Object identifiers identifying the requested Rights Objects, and for each RO identifier an optional hash of the DCF associated with the requested RO. The DCF hash SHOULD be included when the Device is in possession of the associated DCF, unless its inclusion, as determined by some vendor-specific algorithm, would be impractical (e.g. due to the size of the DCF). If the 2-pass protocol is initiated by a ROAP Trigger, the Device SHOULD use the **<contentID>** elements of the ROAP Trigger to identify the associated DCF(s) over which a DCF hash should be calculated. The DCF hash, if computed, MUST be computed as specified in section 5.3 of [DRMDCF-v2.1] using the SHA-1 algorithm.

If the RO refers to more than one (P)DCF, the DRM Agent MAY send multiple DCF Hashes (one per (P)DCF referred by the RO) by duplicating the <roID> in the <roInfo> element. Refer to Annex G.1.6 for an example of the multiple DCF Hashes case.

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-RORequest message:

> *Peer Key Identifier:* An identifier for an RI public key stored in the Device. If the identifier matches the stored RI ID as specified in Section 5.4.2.4.1, 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.
>
> *Transaction Identifier:* Allows a Device to provide the RI with information for tracking of transactions, for example relating to loyalty programs (an example of this could be reward scheme information from the DCF scheme). The Device SHOULD use the **<contentID>** elements of the ROAP Trigger, when present, to identify the associated DCF(s) from which the TransactionID should be extracted. If no **<contentID>** elements have been included in the trigger, then the Transaction Identifier SHOULD not be used.

The Device MUST send the *Peer Key Identifier* extension if, and only if, it has stored the RI public key corresponding to the stored RI ID as specified in Section 5.4.2.4.1. 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 is a signature on this message (besides the *Signature* element itself). The signature method is as follows:

- The message except the *Signature* element is canonicalised according to Section 5.3.3.
- The result of the canonicalisation, *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-RORequest message.

5.4.4.1.2 Message syntax

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

```
<element name="roRequest" type="roap:RORequest"/>

<complexType name="RORequest">
  <annotation>
    <documentation xml:lang="en">
      Message sent from Device to RI to request an RO.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="roap:Request">
      <sequence>
        <element name="deviceID" type="roap:Identifier"/>
        <element name="domainID" type="roap:DomainIdentifier"
                 minOccurs="0"/>
        <element name="riID" type="roap:Identifier"/>
        <element name="nonce" type="roap:Nonce"/>
        <element name="time" type="dateTime"/>
        <element name="roInfo">
          <complexType>
            <sequence maxOccurs="unbounded">
              <element name ="roID" type="ID"/>
```

```
       <element name="dcfHash" minOccurs="0">
        <complexType>
         <sequence>
          <element name="hash" type="base64Binary"/>
         </sequence>
         <attribute name="algorithm" type="anyURI”
                  default="http://www.w3.org/2000/09/xmldsig#sha1"/>
        </complexType>
       </element>
      </sequence>
     </complexType>
    </element>
    <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>
```

The following schema fragment defines the Transaction Identifier extension:

```
<complexType name="TransactionIdentifier">
 <complexContent>
  <extension base="roap:Extension">
   <sequence maxOccurs=”unbounded”>
    <element name="contentID" type=”anyURI”/>
    <element name="id">
     <simpleType>
      <restriction base="string">
       <length value="16"/>
      </restriction>
     </simpleType>
    </element>
   </sequence>
  </extension>
 </complexContent>
</complexType>
```

5.4.4.2 RO Response

The ROAP-ROResponse message is sent from the RI to the Device either in response to a ROAP-RORequest message (two-pass variant) or by RI initiative (one-pass variant). It carries the protected ROs.

5.4.4.2.1 Message description

Parameter	ROAP-ROResponse		
	2-pass *Status = Success*	*2-pass* *Status ≠ Success*	*1-pass*
Status	M	M	M
Device ID	M	-	M
RI ID	M	-	M
Device Nonce	M	-	-
Protected ROs	M	-	M

Certificate Chain	O	-	O
OCSP Response	O	-	M
Extensions	O	-	O
Signature	M	-	M

Table 9: RO Response Message Parameters

Status indicates if the request was successfully handled or not. In the latter case an error code specified in Section 5.3.6 is sent.

Device ID identifies the requesting Device, in the same manner as in the ROAP-DeviceHello message as specified in section 5.4.2.1.1. The value returned here MUST equal the Device ID sent by the Device in the ROAP-RORequest message that triggered this response in the 2-pass ROAP. In the 1-pass ROAP, the value MUST equal the stored Device ID of the recipient Device as defined in Section 5.4.2.4.1. If the Device ID is incorrect, the ROAP-ROResponse processing will fail and the Device MUST discard the received ROResponse 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-RORequest message. In the 1-pass protocol, the value MUST equal the stored RI ID as specified in Section 5.4.2.4.1.

Device Nonce: This parameter, if present (2-pass), MUST have the same value as the corresponding parameter value in the preceding ROAP-RORequest. If the Device Nonce is incorrect, the ROAP-ROResponse processing will fail and the Device MUST discard the received ROResponse PDU.

Protected RO(s) are the Rights Objects (in the form of **<ProtectedRO>** elements), in which sensitive information (such as content encryption keys, CEKs) is encrypted.

Certificate Chain: This parameter MUST be present unless a preceding ROAP-RORequest 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. An RI MUST include this parameter in a 1-pass ROAP if the RI has not received an indication that the Device has cached the RI's certificate chain.

The Device SHOULD check if the RI certificate chain received in this parameter corresponds to stored certificate verification data for this RI. If so, the Device need not verify the RI certificate chain again, otherwise the Device MUST verify the RI certificate chain and MUST compare the hash of the complete DER-encoded subjectPublicKeyInfo component in the received RI certificate with the RI ID from the request. If an RI certificate is received that is not in the stored certificate verification data for this RI, and if 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,` then the Device MUST 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.

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 a preceding ROAP-RORequest (and the RI did not ignore that extension). For the processing of this parameter, see further Section 6.

Extensions: The following extensions are defined for the ROAP-ROResponse message:

> *Transaction Identifier:* Allows an RI to provide a Device with information for tracking of transactions, for example relating to loyalty programs (an example of this could be reward scheme information from the DCF). The RI MUST NOT include a TransactionIdentifier ROAP extension in the ROResponse when the ROResponse contains a RO bound to a GroupID as specified in section 10.7, or a parent ID as defined in section 10.5. Upon reception of a ROResponse containing a TransactionIdentifier ROAP extension and a RO bound to a GroupID a Device MUST ignore the TransactionIdentifier ROAP extension.

> *ConfirmROInstallation*: Indicates to the DRM Agent that it must confirm installation of the ROs contained in this message by sending a ROAP-ROConfirmRequest PDU to the RI. RO Confirmation is a critical extension.

Signature is 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 canonicalised according to Section 5.3.3.
- 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 RO 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 ROs.

Before installing any received ROs that are stateful (indicated by the **stateful** attribute of the <**ro**> element), the Device MUST apply the RO Replay protection described in the Replay Protection Section.

5.4.4.2.2 Message syntax

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

```
<element name="roResponse" type="roap:ROResponse"/>

<complexType name="ROResponse">
  <annotation>
    <documentation xml:lang="en">
      Message sent from RI to Device in response to a roRequest message.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="roap:Response">
      <sequence minOccurs="0">
        <element name="deviceID" type="roap:Identifier"/>
        <element name="riID" type="roap:Identifier"/>
        <element name="nonce" type="roap:Nonce" minOccurs="0"/>
        <element name="protectedRO" type="roap:ProtectedRO" maxOccurs="unbounded"/>
        <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>
```

The **roap:ProtectedRO** type is defined in section 5.3.9.

The following schema fragment defines the *ConfirmROInstallation* extension:

```
<complexType name="ConfirmROInstallation">
  <complexContent>
    <extension base="roap:Extension">
                            <attribute name="sessionId" type="roap:String64" use="required"/>
    </extension>
   </complexContent>
</complexType>
```

Session ID is a protocol session identifier set by the RI. This allows for several, concurrent confirmation sessions. The maximum length of this attribute SHALL be 64 bytes.

The ***critical*** attribute of the *ConfirmROInstallation* type SHALL be present and the value of the attribute SHALL be "true".

5.4.4.3 RO Confirm Request

The ROAP-ROConfirmRequest message is sent from a Device to an RI to confirm installation of one or more Rights Objects delivered in the preceding ROAP-RORResponse. This message SHALL only be sent if the *ConfirmROInstallation* extension was included in the preceding ROAP-ROResponse.

5.4.4.3.1 Message description

ROAP-ROConfirmRequest		
Parameter	***4-pass***	***3-pass***
Session ID	M	M
Device ID	M	M
RI ID	M	M
Device Nonce	M	M
Request Time	M	M
RO Confirm Info	M	M
Certificate Chain	-	O
Extensions	O	O
Signature	M	M

Table 10: RO Confirm Request Message Parameters

Session ID SHALL be identical to the Session ID attribute of the "ConfirmROInstallation" extension parameter of the preceding ROAP-ROResponse message, otherwise the RI SHALL terminate the confirmation protocol.

Device ID identifies the requesting Device. The value MUST equal the stored Device ID.

RI ID identifies the authorizing RI. The value MUST equal the stored RI ID.

Device Nonce is a nonce chosen by the Device. Nonces are generated and used in this message as specified in section 5.3.12.

Request Time is the current DRM Time as measured by the Device. Connected Devices and Unconnected Devices that support DRM Time MUST insert their current DRM Time. Unconnected Devices that do not support DRM Time MUST use the value "**Undefined**".

RO Confirm Info identifies the Rights Object(s) which were previously issued to the Device and their installation status. The parameter consists of a (non-empty) set of Rights Object identifiers and for each RO identifier whether or not the RO was successfully installed by the DRM agent. The status value will be true if the corresponding RO was successfully installed and false if the RO was not installed for whatever reason.

Certificate Chain: This parameter is sent in the 3-pass Confirmed RO Acquisition Protocol 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. This parameter SHALL be omitted for 4-pass Confirmed RO Acquisition Protocol.

Extensions: There are no extensions defined for the ROAP-ROConfirm message.

Signature is a signature on data sent so far in the protocol. The signature is made using the Device's private key on the preceding ROAP-ROResponse and the current message (apart from the *Signature* element itself). The signature method is as follows:

- The previous messages and the current one except the *Signature* element are canonicalised according to Section 5.3.3.
- The two messages are concatenated in their chronological order, starting with the ROAP-ROResponse message. 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 RI MUST verify the signature on the ROAP-ROConfirmRequest message, see section 9.2.

5.4.4.3.2 Message syntax

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

```
<element name="roConfirmRequest" type="roap:ROConfirmRequest"/>
  <complexType name="ROConfirmRequest">
    <annotation>
      <documentation xml:lang="en">
        Message sent from Device to RI to communicate RO installation status.
      </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="time" type="roap:dateTimeOrUndefined"/>
          <element name="roConfirmInfo" type="roap:roConfirmInfo"/>
          <element name="certificateChain" type="roap:CertificateChain" minOccurs="0"/>
          <element name="extensions" type="roap:Extensions" minOccurs="0"/>
          <element name="signature" type="base64Binary"/>
        </sequence>
        <attribute name="sessionId" type="roap:String64" use="required"/>
      </extension>
    </complexContent>
  </complexType>
```

The maximum length of the sessionId attribute SHALL be 64 bytes.

The following schema fragment defines the *RoConfirmInfo* element:

```
<complexType  name="roConfirmInfo">
  <sequence maxOccurs="unbounded">
  <element name ="roID" type="ID"/>
  <element name="installStatus" type="boolean"/>
    </sequence>
 </complexType>
```

5.4.4.4 RO Confirm Response

The ROAP-ROConfirmResponse message is sent from the RI to the Device in response to a ROAP-ROConfirmRequest and completes the confirmation process.

5.4.4.4.1 Message description

Parameter	ROAP-ROConfirmResponse	
	Status = Success	*Status ≠ Success*
Status	M	M
Session ID	M	O
Device Nonce	M	-
RI ID	M	-
Extensions	O	-
Signature	M	-

Table 11: RO Confirm Response Message Parameters

Status indicates if the request was successfully handled or not. In the latter case an error code specified in Section 5.3.6 is sent.

Session ID SHALL be identical to the *Session ID* of the preceding ROAP-ROConfirmRequest (and ROAP-ROResponse) message. If the Session ID of the ROAP-ROConfirmResponse does not equal the Session ID of the corresponding ROAP-ROConfirmRequest, the Device MUST terminate the protocol. . The Session ID can be present only if the Rights Issuer could detect the session identifier in the ROAP-ROConfirmRequest.

Device Nonce is the nonce sent by the Device. This parameter MUST have the same value as the corresponding parameter value in the preceding ROAP-ROConfirmRequest. If the Device Nonce is incorrect, the ROAP-RoConfirmResponse processing will fail and the Device MUST discard the received RO Confirm Response PDU.

RI ID identifies the RI. The value MUST equal the RI ID sent by the Device in the preceding ROAP-ROConfirmRequest message.

Extensions: No extensions are defined for the ROAP-ROConfirmResponse message.

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

- The current one except the *Signature* element is canonicalised according to Section 5.3.3.
- The signature is calculated on the canonicalised message in accordance with the rules of the negotiated signature scheme

The Device MUST verify this signature. A Device MUST NOT accept the confirmed RO Acquisition protocol as successful unless the signature verifies.

5.4.4.4.2 Message syntax

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

```
<element name="roConfirmResponse" type="roap:ROConfirmResponse"/>

<complexType name="ROConfirmResponse">
  <annotation>
    <documentation xml:lang="en">
      Message sent from RI to Device in response to a roConfirmRequest message.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="roap:Response">
```

```
   <sequence minOccurs="0">
     <element name="nonce" type="roap:Nonce"/>
     <element name="riID" type="roap:Identifier"/>
     <element name="extensions" type="roap:Extensions" minOccurs="0"/>
     <element name="signature" type="base64Binary"/>
   </sequence>
   <attribute name="sessionId" type="roap:String64" />
  </extension>
 </complexContent>
</complexType>
```

The maximum length of the sessionId attribute SHALL be 64 bytes.

5.4.5 Domain Management

5.4.5.1 Join Domain Request

The ROAP-JoinDomainRequest message is sent from a Device to an RI and is a request to join a Domain. This message is the first of the 2-pass Join Domain protocol.

5.4.5.1.1 Message description

ROAP-JoinDomainRequest	
Parameter	**Mandatory/Optional**
DeviceID	M
RI ID	M
Device Nonce	M
Request Time	M
Domain Identifier	M
Certificate Chain	O
Extensions	O
Signature	M

Table 12: Join Domain Request Message Parameters

Device ID identifies the requesting Device. The value MUST equal the stored Device ID as specified in Section 5.4.2.4.1.

RI ID identifies the authorizing RI. The value MUST equal the stored RI ID as specified in Section 5.4.2.4.1.

Device Nonce is a nonce chosen by the Device. Nonces are generated and used in this message as specified in section 5.3.12.

Request Time is the current DRM Time, as seen by the Device. Connected Devices and Unconnected Devices that support DRM Time MUST insert their current DRM Time. Unconnected Devices that do not support DRM Time MUST use the value "**Undefined**".

Domain Identifier shall identify the Domain the Device wishes to join.

Certificate Chain: This parameter is sent unless *Certificate Caching* is indicated in the RI Context with this RI. 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-JoinDomainRequest message:

Peer Key Identifier: An identifier for an RI public key stored in the Device. If the identifier matches the stored RI ID as specified in Section 5.4.2.4.1 or if it 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.

Hash Chain Support: When this extension is present, it signals that the client supports a technique of generating Domain Keys through hash chains, see section 8.8.1.

The Device MUST send the *Peer Key Identifier* extension if, and only if, it has stored the RI public key corresponding to the stored RI ID as specified in Section 5.4.2.4.1. 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’s 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. The Device MUST send the *Hash Chain Support* extension if, and only if, it supports hash-chained Domain keys.

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

- The message except the *Signature* element is canonicalised according to Section 5.3.3.
- The result of the canonicalisation, *d*, is considered as input to the signature operation. The signature is calculated on *d* in accordance with the rules of the negotiated signature algorithm.

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

5.4.5.1.2 Message syntax

The <**joinDomainRequest**> element specifies the ROAP-JoinDomainRequest message. It has complex type **roap: DomainRequest**, which extends the basic **roap:Request** type. Note that this type is used both for join and leave Domain request messages.

```
<element name="joinDomainRequest" type="roap:DomainRequest"/>

<complexType name="DomainRequest">
 <annotation>
  <documentation xml:lang="en">
   General PDU for sending Domain-related requests from a Device to an RI.
  </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="time" type="roap:dateTimeOrUndefined"/>
    <element name="domainID" type="roap:DomainIdentifier"/>
    <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>
```

The following schema fragment defines the **roap:DomainIdentifier** type. The last three characters (digits) represent the Domain Generation (see section 8.8 for further information). The other, preceding characters represent the Domain baseID.

RIs will always respond with the Domain Key corresponding to the most recent Domain Generation and, if hash chains are not supported, all earlier Domain Keys for this Domain too.

```
<simpleType name="DomainIdentifier">
  <restriction base="string">
    <pattern value=".{1,17}\d{3}"/>
  </restriction>
</simpleType>
```

The following schema fragment defines the Hash Chain Support extension:

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

5.4.5.2 Join Domain Response

The ROAP-JoinDomainResponse message is sent by an RI to a Device in response to a ROAP-JoinDomainRequest message. This message is the second message in the 2-pass protocol to join a Device to a Domain.

5.4.5.2.1 Message description

Parameter	**ROAP-JoinDomainResponse**	
	Status = "Success"	***Status ≠ "Success"***
Status	M	M
Device ID	M	-
RI ID	M	-
Device Nonce	M	-
Domain Info	M	-
Certificate chain	O	-
OCSP Response	O	-
Extensions	O	-
Signature	M	-

Table 13: Join Domain Response Message Parameters

Status indicates if the request was successfully handled or not. In the latter case an error code as specified in Section 5.3.6 is sent.

Device ID identifies the requesting Device. The value returned here MUST equal the Device ID sent by the Device in the ROAP-JoinDomainRequest message that triggered this response.

RI ID identifies the RI. The value returned here MUST equal the RI ID sent by the Device in the preceding ROAP-JoinDomainRequest message.

Device Nonce: This parameter MUST have the same value as the corresponding parameter value in the preceding ROAP-JoinDomainRequest. If the Device Nonce is incorrect, the ROAP-Join Domain Response processing will fail and the Device MUST discard the received Join Domain Response PDU.

Domain Info: This parameter carries Domain keys (encrypted using Device's public key) as well as information about the maximum lifetime of the Domain. Devices MAY use a shorter lifetime than suggested by the RI.

Certificate Chain: This parameter MUST be present unless a preceding ROAP-JoinDomainRequest 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.

The Device SHOULD check if the RI certificate chain received in this parameter corresponds to stored certificate verification data for this RI. If so, the Device need not verify the RI certificate chain again, otherwise the Device MUST verify the RI certificate chain and MUST compare the hash of the complete DER-encoded subjectPublicKeyInfo component in the received RI certificate with the RI ID from the request. If an RI certificate is received that is not in the stored certificate verification data for this RI, and if 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," then the Device MUST 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.

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-JoinDomainRequest (and the RI did not ignore that extension). For the processing of this parameter, see further Section 6.

Extensions: The following extension is currently defined for the ROAP-JoinDomainResponse message:

> *Hash Chain Support*: When this extension is present it indicates that the RI is using the technique of generating Domain Keys through hash chains described in the Domains Section. The RI MUST NOT include this extension in the ROAP-JoinDomainResponse unless the same extension was received in the preceding ROAP-JoinDomainRequest. If the Device receives the Hash Chains Support extension then it needs only store the latest Domain Key for a given Domain.

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

- The message except the *Signature* element is canonicalised according to Section 5.3.3.
- The result of the canonicalisation, *d*, is considered as input to the signature operation The signature is calculated on *d* in accordance with the rules of the negotiated signature algorithm.

The Device MUST verify this signature. A Device MUST NOT accept the Join Domain protocol 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 Join Domain protocol failed the Device MUST NOT store a Domain Context, otherwise the Device MUST store the resulting Domain Context.

The stored Domain Context SHALL at a minimum contain: The Domain ID (which includes the Domain Generation), the Domain Context Expiry Time, and, if applicable, an indication that the RI supports hash-chained Domain Keys. . If the process of joining a Domain has started with a Domain Trigger that contained the **<domainAlias>** element, the Domain Context SHALL also contain the Domain Alias. If the Device and RI both support hash chains, the Domain Context SHALL contain the Domain Key corresponding to the highest known generation, otherwise the Domain Context SHALL contain all Domain Keys of all Domain Generations. As Domain ROs are signed the DRM Agent needs to have the RI Public Key to validate distributed Domain ROs. Devices supporting Domains MUST store the RI Public Key in either the Domain Context or the RI Context.

A Device MUST NOT install any Domain ROs for a Domain whose Domain Context has expired. In the case of Unconnected Devices that do not support DRM Time, the Domain Context does not expire and hence has a value that is infinite, as indicated in the **DomainInfo:NotAfter** element.

NOTE: Rights Issuers should carefully consider the security implications of using the value “Infinite” for Devices that support DRM Time.

A Device MAY have several Domain Contexts with an RI.

5.4.5.2.2 Message syntax

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

```
<element name="joinDomainResponse" type="roap:JoinDomainResponse"/>

<complexType name="JoinDomainResponse">
  <annotation>
    <documentation xml:lang="en">
      Message sent from RI to Device in response to a joinDomainRequest.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="roap:Response">
      <sequence minOccurs="0">
        <element name="deviceID" type="roap:Identifier"/>
        <element name="riID" type="roap:Identifier"/>
        <element name="nonce" type="roap:Nonce"/>
        <element name="domainInfo" type="roap:DomainInfo"/>
        <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>
```

The following schema fragment defines the **DomainInfo** type and <extendedDomainInfo> element::

```
<complexType name="DomainInfo">
  <sequence>
    <element name="notAfter" type="roap:dateTimeOrInfinite"/>
        <element name="domainKey" type="roap:ProtectedDomainKey" maxOccurs="unbounded"
form="qualified"/>
  </sequence>
</complexType>

<element name="extendedDomainInfo">
  <complexType>
    <sequence>
      <element name="noConsumeAfter" type="roap:dateTimeOrInfinite"/>
        <any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>


<simpleType name="dateTimeOrInfinite">
  <union memberTypes="dateTime roap:InfiniteString"/>
</simpleType>

<simpleType name="InfiniteString">
  <restriction base="string">
  <enumeration value="Infinite"/>
  </restriction>
</simpleType>
```

The <**notAfter**> element expresses, in UTC, the expiry time of the Domain Context. The value **”Infinite”** indicates infinite lifetime of the Domain Context.

The <**extendedDomainInfo**> element MAY be used to express additional characteristics of the Domain. If used, then the <**extendedDomainInfo**> MUST be located inside an extension of type **ExtensionContainer** in the <**extensions**>**-**element of the <**joinDomainResponse**>**.**

The <**noConsumeAfter**> element expresses, in UTC, a time after which the DRM Agent MUST NOT grant any permissions from Rights Objects that have been issued for this Domain. The value **”Infinite”** indicates that normal evaluation of installed Rights Objects is always allowed.

Future versions of OMA DRM MAY define additional Domain related information to be located in the <**extendedDomainInfo**> element after the <**noConsumeAfter**> element (validating against the <**any/**> element). Unknown elements MUST be disregarded.

The <**domainKey**> element contains the wrapped Domain key and a key-confirming MAC key, see below.

```
<complexType name="ProtectedDomainKey">
  <sequence>
    <element name="encKey" type="xenc:EncryptedKeyType"/>
    <element name="riID" type="roap:Identifier"/>
    <element name="mac" type="base64Binary"/>
  </sequence>
</complexType>
```

The <**encKey**> element contains a MAC key, K_{MAC}, and a Domain Key, K_D, wrapped as specified in the Key Management section 7. The value of the <**encKey**> element's **Id** attribute must equal the value of the <**domainID**> element in the preceding ROAP-JoinDomainRequest message, save for the Domain Generation part. If Hash Chains are supported by both the Device and the RI, only the Domain Key corresponding to the most recent Domain Generation SHOULD be included, otherwise all Domain Keys for all Domain Generations MUST be included (including their domain identifiers as **Id** attributes). The child of the <**ds:KeyInfo**> element inside the <**encKey**> element SHALL be the <**roap:X509SPKIHash**> element, identifying a particular DRM Agent's public key through the hash of the `subjectPublicKeyInfo` value in its certificate.

The <**riID**> element is necessary for key confirmation purposes. A Device MUST verify that it has the same value as the <**riID**> element of the ROAP-JoinDomainResponse message itself.

The <**mac**> element provides key-confirmation through a MAC on the canonical version according to Section 5.3.3 of the <**domainKey**> element (excluding the <**mac**> element itself) using the MAC key K_{MAC} wrapped in the <**encKey**> element. The MAC algorithm to use is defined by the RI Context. Devices MUST NOT install Domain keys where the MAC is invalid.

5.4.5.3 Leave Domain Request

The ROAP-LeaveDomainRequest message is sent from the Device to the RI. This message is the first message in the 2-pass protocol for removing a Device from a Domain.

5.4.5.3.1 Message description

ROAP-LeaveDomainRequest	
Parameter	**Mandatory/Optional**
DeviceID	M
RI ID	M
Device Nonce	M

Request Time	M
Domain Identifier	M
Certificate Chain	O
Extensions	O
Signature	M

Table 14: Leave Domain Request Message Parameters

Device ID identifies the requesting Device. The value MUST equal the stored Device ID as specified in Section 5.4.2.4.1.

RI ID identifies the authorizing RI. The value MUST equal the stored RI ID as specified in Section 5.4.2.4.1.

Device Nonce is a nonce chosen by the Device. Nonces are generated and used in this message as specified in section 5.3.12.

Request Time is the current DRM Time, as seen by the Device. Connected Devices and Unconnected Devices that support DRM Time MUST insert their current DRM Time. Unconnected Devices that do not support DRM Time MUST use the value "**Undefined**".

Domain Identifier identifies the Domain.

Certificate Chain: This parameter is sent unless *Certificate Caching* is indicated in the RI Context with this RI. When present, the parameter value shall be as described for the *Certificate Chain* parameter in the ROAP-RegistrationRequest message.

Extensions: The following extension is currently defined for the ROAP-LeaveDomainRequest message:

> *Not a Domain Member*: Presence of this extension indicates to the RI that the Device does not consider itself a member of this Domain (even though it is sending a request for the RI to remove it from the Domain). This could happen, for example, if the Device already has left the Domain, but receives a new trigger to leave it (perhaps because the RI never received the previous ROAP-LeaveDomainRequest). This extension MUST be included in the request if the Device is not a member of the identified Domain.

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

- The message except the *Signature* element is canonicalised according to Section 5.3.3.
- The result of the canonicalisation, *d*, is considered as input to the signature operation. The signature is calculated on *d* in accordance with the rules of the negotiated signature algorithm.

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

The Device MUST ensure that the Domain Context of the corresponding Domain is deleted **before** sending the ROAP-LeaveDomainRequest to the RI.

5.4.5.3.2 Message syntax

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

```
<element name="leaveDomainRequest" type="roap:DomainRequest"/>
```

The following schema fragment defines the Not a Domain Member extension:

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

5.4.5.4 Leave Domain Response

The ROAP-LeaveDomainResponse message is sent by an RI to a Device in response to a ROAP-LeaveDomainRequest message. This message is the second message in the 2-pass protocol for removing a Device from a Domain.

5.4.5.4.1 Message description

ROAP-LeaveDomainResponse		
Parameter	**Mandatory/Optional**	
	Status = "Success"	**Status ≠ "Success"**
Status	M	M
Device Nonce	M	-
Domain Identifier	M	-
Extensions	O	-

Table 15: Leave Domain Response Message Parameters

Status indicates if the request was successfully handled or not. In the latter case an error code defined in section 5.3.6 is sent.

Device Nonce is the nonce sent by the Device. This parameter MUST have the same value as the corresponding parameter value in the preceding ROAP-LeaveDomainRequest. If the Device Nonce is incorrect, the ROAP-Leave Domain Response processing will fail and the Device MUST discard the received Leave Domain Response PDU.

Domain Identifier identifies the Domain from which the RI removed the Device. The Domain Generation part of the Domain Identifier SHALL be ignored.

Extensions: No extensions are currently defined for the ROAP-LeaveDomainResponse message.

The RI sends the ROAP-LeaveDomainResponse after having deleted the association of this Device to the Domain (i.e. updated the Domain membership status).

5.4.5.4.2 Message Syntax

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

```
<element name="leaveDomainResponse" type="roap:LeaveDomainResponse"/>

<complexType name="LeaveDomainResponse">
  <annotation>
    <documentation xml:lang="en">
      Message sent from RI to Device in response to a leaveDomainRequest
    </documentation>
  </annotation>
  <complexContent>
    <extension base="roap:Response">
      <sequence minOccurs="0">
        <element name="nonce" type="roap:Nonce"/>
        <element name="domainID" type="roap:DomainIdentifier"/>
        <element name="extensions" type="roap:Extensions" minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

5.4.6 Metering Report

5.4.6.1 MeteringReportSubmit

The ROAP-MeteringReportSubmit message is sent from a Device to an RI to send a Metering Report. This message is the first message of the 2-pass Metering Report protocol.

5.4.6.1.1 Message description

ROAP-MeteringReportSubmit	
Parameter	**Mandatory/Optional**
Device ID	M
RI ID	M
Device Nonce	M
Report Time	M
MeteringReport	M
Certificate Chain	O
Extensions	O
Signature	M

Table 16: Metering Report Submit Message Parameters

Device ID identifies the requesting Device. The value MUST equal the stored Device ID as specified in Section 5.4.2.4.1.

RI ID identifies the RI. The value MUST equal the stored RI ID as specified in Section 5.4.2.4.1.

Device Nonce is a nonce chosen by the Device. Nonces are generated and used in this message as specified in section 5.3.12.

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

MeteringReport: contains the encrypted Metering Report.

Certificate Chain: This parameter is sent unless it is indicated in the RI Context that this RI has stored necessary Device certificate information.

Extensions:

> *Peer Key Identifier:* An identifier for an RI public key stored in the Device. If the identifier matches the stored RI ID as specified in Section 5.4.2.4.1, 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.

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

- The message except the *Signature* element is canonicalised according to Section 5.3.3.
- The result of the canonicalisation, *d*, is considered as input to the signature operation.

The RI MUST verify the signature on the ROAP-MeteringReportSubmit message. A RI MUST NOT accept the Metering Information stored in a Metering Report if it can not verify the signature,

5.4.6.1.2 Message syntax

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

```
<element name="meteringReportSubmit" type="roap:MeteringReportSubmit"/>
<complexType name="MeteringReportSubmit">
  <annotation>
    <documentation xml:lang="en">
      Message sent from Device to RI to submit a metering 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="time" type="dateTime"/>
        <element name="meteringReport">
          <complexType>
            <sequence>
              <element name="encryptedMeteringReport" type="xenc:EncryptedDataType”/>
              <element name="encKey" type="xenc:EncryptedKeyType"/>
              <element name="mac" type="base64Binary"/>
            <sequence>
          </complexType>
        </element>
        <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>
```

The <**deviceID**> element is of type **roap:Identifier** and SHALL identify the issuing Device.

The <**encKey**> element of the <**meteringReport**> element is of type **xenc:EncryptedKeyType** from [XML-Enc]. It consists of a wrapped concatenation of a MAC key, K_{MAC} and a Metering Encryption Key, K_{MEK}. 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 the <**ds:KeyInfo**> element inside the <**encryptedMeteringReport**> element. The <**ds:KeyInfo**> child element of the <**encKey**> element SHALL identify the wrapping key. For further information on packaging the MAC key and the Metering Encryption key, see the Key Management discussion in section 7. The <**ds:KeyInfo**> element SHALL be the <**roap:X509SPKIHash**> element, identifying the RIs Public Key through the (SHA-1) hash of the DER-encoded subjectPublicKeyInfo value in the RIs Certificate

The <**mac**> element MUST be present. The <**mac**> element provides key-confirmation through a MAC on the canonical version according to Section 5.3.3 of the <**meteringReport**> element (excluding the <**mac**> element) using the MAC key K_{MAC} wrapped in the <**encKey**> element, the MAC algorithm SHALL be the same algorithm that was negotiated as part of the registration with the RI i.e. the MAC algorithm stored in the RI Context.

The following schema fragment defines the **meteringReportType**, which holds the Full Metering Report (formatted as defined in section 11.4) and SHALL be encrypted using [XML-Enc], the resultant encrypted data will be present in the <**encryptedMeteringReport**> element of the MeteringReportSubmit message.

```
<complexType name="meteringReportType">
 <sequence>
  <element name="rawMeteringReportData" type="string"/>
  <any processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
 <sequence>
</complexType>
```

Future versions of OMA DRM MAY define additional metering information to be located in the <meteringReportType> after the <**rawMeteringReportData**> element (validating against the <**any**/> element). Unknown elements MUST be disregarded.

5.4.6.2 MeteringReportResponse

The ROAP-MeteringReportResponse message is sent from the RI to the Device in response to a ROAP-MeteringReportSubmit message.

5.4.6.2.1 Message description

Parameter	ROAP-MeteringReportResponse
Status	M
Device ID	M
RI ID	M
Device Nonce	M
Certificate Chain	O
OCSP Response	O
Extensions	O
Signature	M

Table 17: Metering Report Response Message Parameters

Status indicates if the request was successfully handled or not. In the latter case an error code specified in Section 5.3.6 is sent.

Device ID identifies the requesting Device, in the same manner as in the ROAP-DeviceHello message as specified in section 5.4.2.1.1. The value returned here MUST equal the Device ID sent by the Device in the ROAP-MeteringReportSubmit message that triggered this response.

RI ID identifies the RI. The value MUST equal the RI ID sent by the Device in the preceding ROAP-MeteringReportSubmit message.

Device Nonce: This parameter MUST have the same value as the corresponding parameter value in the preceding ROAP-MeteringReportSubmit. If the Device Nonce is incorrect, the ROAP-MeteringReportResponse processing will fail and the Device MUST discard the received MeteringReportResponse PDU.

Certificate Chain: This parameter MUST be present unless a preceding ROAP-MeteringReportSubmit 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.

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-MeteringReportSubmit (and the RI did not ignore that extension). For the processing of this parameter, see further Section 6.

Extensions: The following extensions are currently defined for the ROAP-MeteringReportResponse message:

> *Post Response URL*: This allows an RI to bind a subsequent RO acquisition or other ROAP session to the execution of the Metering Report protocol.

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

- The message except the *Signature* element is canonicalised according to Section 5.3.3.
- The result of the canonicalisation, *d*, is considered as input to the signature operation. The signature is calculated on *d* in accordance with the rules of the negotiated signature algorithm.

The Device MUST verify this signature. A Device MUST NOT accept the Metering Report protocol 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.

5.4.6.2.2 Message syntax

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

```
<element name="meteringReportResponse" type="roap:MeteringReportResponse"/>

<complexType name="MeteringReportResponse">
  <annotation>
    <documentation xml:lang="en">
      Message sent from RI to Device in response to a meteringReportSubmit message.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="roap:Response">
      <sequence minOccurs="0">
        <element name="deviceID" type="roap:Identifier"/>
        <element name="riID" type="roap:Identifier"/>
        <element name="nonce" type="roap:Nonce"/>
        <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>
```

The following schema fragment defines the *Post Response URL* extension:

```
<complexType name="postResponseURL">
  <complexContent>
    <extension base="roap:Extension">
      <sequence>
        <element name="prURL" type=”anyURI”/>
      <sequence>
    </extension>
  </complexContent>
</complexType>
```

prURL specifies a URL. The value of the **<prURL>** element MUST be a URL according to [RFC2396]. A successful request to the URL MUST return a ROAP Trigger, a Download Descriptor or a bundled Download Descriptor and ROAP Trigger. The processing of the <prURL> element is as described in section 11.3.

The ***critical*** attribute SHALL be present and the value of the attribute SHALL be “true” and this extension MUST be supported by the Device.

5.4.7 RO Upload

5.4.7.1 RO Upload Request

The ROAP-ROUploadRequest message is sent from a Device to an RI to upload Rights Objects. If the RO being uploaded is stateful the Device MUST report the current state information to the RI. Before sending this message, the Device MUST disable the ROs being uploaded..

5.4.7.1.1 Message description

ROAP-RO UploadRequest	
Parameter	**Mandatory/Optional**
Device ID	M
RI ID	M
Device Nonce	M
Request Time	M
RO Info	M
Certificate Chain	O
Extensions	O
Signature	M

Table 18: RO Network Backup Request Message Parameters

Device ID identifies the requesting Device. The value MUST equal the stored Device ID as specified in Section 5.4.2.4.1.

RI ID identifies the authorizing RI. The value MUST equal the stored RI ID as specified in Section 5.4.2.4.1.

Device Nonce is a nonce chosen by the Device. Nonces are generated and used in this message as specified in section 5.3.12.

Request Time is the current DRM Time, as seen by the Device.

RO Info identifies the ROs being uploaded and their current state information.

Certificate Chain: This parameter is sent unless *Certificate Caching* is indicated in the RI Context with this RI. 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-ROUploadRequest message:

> *Peer Key Identifier:* An identifier for an RI public key stored in the Device. If the identifier matches the stored RI ID as specified in Section 5.4.2.4.1 or if it 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 corresponding to the stored RI ID as specified in Section 5.4.2.4.1. 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's 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 is a signature on this message (besides the *Signature* element itself). The signature method is as follows:

- The message except the *Signature* element is canonicalised according to Section 5.3.3.
- The result of the canonicalisation, *d*, is considered as input to the signature operation.

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

5.4.7.1.2 Message syntax

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

```
<element name="roUploadRequest" type="roap:ROUploadRequest"/>
<complexType name="ROUploadRequest">
  <annotation>
    <documentation xml:lang="en">
      Message sent from Device to RI to upload ROs.
    </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="time" type="dateTime"/>
        <element name="roInfo">
          <complexType>
            <sequence maxOccurs="unbounded">
             <element name="roID" type="ID"/>
             <element name="stateInfo" type="o-ex:constraintType" minOccurs="0"
maxOccurs="unbounded"/>
            </ sequence >
           </complexType>
        </element>
        <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>
```

The <roID> element identifies the original RO issued by the RI. The value equals the “id” attribute of the <ro> element in the <protectedRO> element issued by the RI.

The <stateInfo> element is of type o-ex:constraintType [ODRL] and used to express the current state information of the RO to be uploaded . The <stateInfo> MUST be repeated for every <constraint> in the original <ro> element that contains an ”id” attribute. Only <constraint> elements with an “id” attribute are reported. If the RI supports the Rights Object Upload protocol, then it MUST add the “id” attribute to any of the stateful constraints for all ROs it issues, see [DRMREL-v2.1]. A stateful constraint is a <constraint> element that contains one of the following elements: <count>, <timed-count>, <interval> or <accumulated>. For the <count> and <timed-count> elements, the value contains the remaining count value. For the <accumulated> element, the value contains the remaining duration that the Content can be rendered (in the format of the <accumulated> element). The <interval> element is handled differently. If the Content has not been rendered, i.e. the interval has not started, then nothing is placed in the <stateInfo> element. If the Content has been rendered, i.e. the interval has been started, then the <interval> is transformed into a <datetime><end>*xx*</end></datetime>, where *xx* is the end date/time of after which the Content can not be rendered.

5.4.7.2 RO Upload Response

The ROAP-ROUploadResponse message is sent from the RI to the Device in response to a ROAP-ROUploadRequest message. This message is the second message in the 2-pass protocol to upload RO.

5.4.7.2.1 Message description

Parameter	ROAP-RO UploadResponse
Status	M
Device ID	M
RI ID	M
Device Nonce	M
Certificate chain	O
OCSP Response	O
Extensions	O
Signature	M

Table 19: RO UploadResponse Message Parameters

Status indicates if the request was successfully handled or not. In the latter case an error code as specified in Section 5.3.6 is sent.

Device ID identifies the requesting Device. The value returned here MUST equal the Device ID sent by the Device in the ROAP-ROUploadRequest message that triggered this response.

RI ID identifies the RI. The value returned here MUST equal the RI ID sent by the Device in the preceding ROAP-ROUploadRequest message.

Device Nonce: This parameter MUST have the same value as the corresponding parameter value in the preceding ROAP-ROUploadRequest. If the Device Nonce is incorrect, the ROAP-RO UploadResponse processing will fail and the Device MUST discard the received RO UploadResponse PDU.

Certificate Chain: This parameter MUST be present unless a preceding ROAP-ROUploadRequest 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-ROUploadResponse message.

The Device SHOULD check if the RI certificate chain received in this parameter corresponds to stored certificate verification data for this RI. If so, the Device need not verify the RI certificate chain again, otherwise the Device MUST 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 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," then the Device MUST 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.

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-ROUploadRequest (and the RI did not ignore that extension). For the processing of this parameter, see further Section 6.

Extensions: The current extensions are defined for ROAP-ROUploadResponse message.

> *Fail:* The presence of this extension indicates that one or more ROs failed to be uploaded. If present, the extension will list one or more RO IDs from the original RO Upload Request which failed to upload, and optionally, the RO IDs can be accompanied with a corresponding reason indicates why the RO(s) failed to be uploaded.. This extension only exists when the status equals to "InvalidUploadedRO" or "UnknownUploadedRO".

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

- The message except the *Signature* element is canonicalised according to Section 5.3.3.
- The result of the canonicalisation, *d*, is considered as input to the signature operation

The Device MUST verify this signature. A Device MUST NOT accept the RO Upload protocol 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 RO Upload protocol failed the Device MUST enable the ROs that it was attempting to upload, otherwise the Device MUST permanently remove the ROs that it was attempting to upload .

5.4.7.2.2 Message syntax

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

```
<element name="roUploadResponse" type="roap:ROUploadResponse"/>

<complexType name="ROUploadResponse">
  <annotation>
    <documentation xml:lang="en">
      Message sent from RI to Device in response to a ROUploadRequest.
    </documentation>
  </annotation>
  <complexContent>
    <extension base="roap:Response">
      <sequence minOccurs="0">
        <element name="deviceID" type="roap:Identifier"/>
        <element name="riID" type="roap:Identifier"/>
        <element name="nonce" type="roap:Nonce"/>
        <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>
```

The following schema fragment defines the *Fail*extension:

```
<complexType name="Fail">
  <complexContent>
    <extension base="roap:Extension"/>
        <sequence minOccurs=”0”>
          <element name="roID" type="ID" maxOccurs="unbounded"/>
         <element name="reason" minOccurs="0">
```

```
            <simpleType>
              <restriction base="string">
                <maxLength value="256"/><!--the unit is ‘byte’-->
              </restriction>
            </simpleType>
          </element>
        </sequence>
    </extension>
  </complexContent>
</complexType>
```

6. Certificate Status Checking & Device Time Synchronisation

6.1 Certificate status checking by RI

For each request signed by the Device that requires the RI to perform substantial or security-related processing, the RI MUST check the signature, expiry date (validity), and the revocation status of all certificates in the Device certificate chain.

6.2 Certificate status checking by DRM Agents

A Device MUST verify signed RI responses and ROs. The signature verification MUST include a check of the validity of all the certificates in the RI certificate chain, and of the revocation status of all revocable certificates in the RI certificate chain, with the exception that in the Domain RO installation process, revocation status check MAY be omitted as specified in 8.7.2.1. To allow the Device to do the certificate status check, the RI MUST include OCSP responses for all revocable certificates in the RI certificate chain when sending signed responses to the Device. The only exception to this is when the Device has sent the No OCSP Response extension in the request that triggered the RI response. In case of a ROAP-RegistrationResponse containing a nonce-based OCSP response the Device MUST first process the OCSP response as specified in 6.3. The determination of which certificates in an RI certificate chain are revocable is deemed to be part of the trust model of the root of trust of that chain. In case the root of trust does not specify such a policy, Devices SHALL assume a default model. In the default model only the RI certificate is revocable and requires an OCSP response to prove its status.

A Device which did not send the No OCSP Response extension in its ROAP-Request message MUST check that an OCSP response is present in the received ROAP-Response message. If no OCSP response is present then the Device MUST abort the protocol.

When providing OCSP responses to Devices that do support DRM time, the RI MAY disregard whether a nonce is present in an OCSP response or not. The exception to this is when the RI deems the Device's time to be out of sync during Registration, see further Section 6.3.

To reduce the load on OCSP responders, RIs SHOULD use locally cached OCSP responses to the extent possible. However, per [OCSP-MP], if an OCSP response does not have the nextUpdate present, then the RI MUST NOT cache the OCSP response.

Unconnected Devices that do not support DRM Time will not be able to use time-based OCSP responses. Because of this, RIs SHOULD only use nonce-based OCSP responses (with the nonce supplied by the Device) when communicating with Unconnected Devices that do not support DRM Time.

The Device MUST verify that the OCSP-provided status of all revocable certificates in the RI certificate chain is `good`. A Device MUST be able to detect that an OCSP responder certificate is non-revocable through the use of the `id-pkix-ocsp-nocheck` extension (see further Appendix E).

DRM Agents MUST support all client requirements in [OCSP-MP] with the following exceptions:

DRM Agents need not be able to generate OCSP requests

DRM Agents need only be able to handle OCSP responses with one `SingleResponse` value

DRM Agents need not support the `authorityInfoAccess` certificate extension (as they will not contact OCSP responders directly)

DRM Agents need not support OCSP over HTTP/1.1 (as they will not contact OCSP responders directly)

Devices MUST be able to match a nonce sent for OCSP purposes in the ROAP protocol with a nonce in the received OCSP response.

6.3 Device DRM Time Synchronisation

An RI, which receives a ROAP-RORequest or a ROAP-JoinDomainRequest or a ROAP-MeteringReportSubmit, and detects that the Device's DRM Time as specified in the request is inaccurate, SHALL respond with the status code DeviceTimeError. A Device receiving this status code SHOULD attempt to re-register with the RI by initiating the Registration protocol.

An RI, which receives a ROAP-RegistrationRequest, and detects that the Device's time as specified in the request is inaccurate, MUST send an OCSP request to its responder, and include the nonce sent by the Device in the OCSP request. The nonce-based OCSP response returned from the OCSP responder MUST be included in the RegistrationResponse message sent back to the Device.

To ensure DRM time synchronisation interoperability, Rights Issuers MUST place the value of the nonce contained within the ROAP-RegistrationRequest into the OCSP extnValue structure as a DER encoded OCTET STRING.

For example, if the octet representation of the value of the nonce contained within the ROAP-RegistrationRequest is “9C ED F0 3F 24 D7 72 8D 57 C6 C7 44 D1 07 72 F8”, then the proper encoding of this value into the OCSP extnValue structure is as described in the following table.

➢ ASN.1	➢ DER Encoding (HEX)
➢ SEQUENCE {	➢ 30 1F
OBJECT IDENTIFIER	06 09
ocspNonce (1 3 6 1 5 5 7 48 1 2)	2B 06 01 05 05 07 30 01 02
OCTET STRING, encapsulates {	04 12
OCTET STRING	04 10
9C ED F0 3F 24 D7 72 8D 57 C6 C7 44 D1 07 72 F8	9C ED F0 3F 24 D7 72 8D 57 C6 C7 44 D1 07 72 F8
}	
}	

A Device, which receives a ROAP-RegistrationResponse message containing a nonce-based OCSP response where the nonce in the OCSP response matches the nonce sent in the Device's ROAP-RegistrationRequest, MUST validate the OCSP response and the expiry time of all certificates from the OCSP responders certificate chain using the time in the `producedAt` component of the OCSP response. Assuming this was successful, the Device MUST adjust the DRM Time for the current trust model to the time in the producedAt component of the OCSP response. The validation of the RegistrationResponse (and of the Rights Issuers certificate expiry times) shall be performed afterwards by using this DRM Time. Unconnected Devices that do not support DRM Time SHALL also use this time to validate the RegistrationResponse and may forget it afterwards. Barring network latency and response times, the procedure described here will synchronize the Device’s DRM Time with the OCSP responder's.

To avoid excessive re-registrations and a high load on OCSP responders,

Rights Issuers MUST use the time obtained from the OCSP responders as its reference time in order to judge the inaccuracies in the Device’s DRM Time.

Rights Issuers SHOULD allow for a reasonable drift in the Device’s DRM Time.

Connected Devices and Unconnected Devices that support DRM Time should maintain DRM Time to an accuracy of 120ppm (this equates to approximately 60 minutes per year).

Connected Devices MUST support DRM Time.

Unconnected Devices are RECOMMENDED to support DRM Time. An Unconnected Device might not support DRM Time because it is considered too onerous for a limited functionality Device, but, in order to maximize the security of the overall OMA DRM Version 2 system, implementers are encouraged to implement Unconnected Devices supporting DRM Time whenever possible.

7. Key Management

7.1 Cryptographic Components

7.1.1 RSAES-KEM-KWS

RSA-KEM-KWS is an asymmetric encryption scheme defined in [X9.44] and based on the "generic hybrid cipher" in [ISO/IEC 18033]. In this scheme, the sender uses the recipient's public key to securely transfer symmetric-key material to the recipient. Specifically, given the recipient's public RSA key P=(m,e), consisting of a modulus m and a public exponent e, the sender generates a value Z as a statistically uniform random integer in the interval [0,…,m-1]. The value Z is then converted to a key-encryption key KEK as follows:

$$KEK = \mathrm{KDF}(\mathrm{I2OSP}(Z, mLen)\ \boldsymbol{NULL},\ kekLen)$$

where KDF is defined below, I2OSP converts a nonnegative integer to an octet string of a specified length and is defined in [PKCS-1], *mLen* is the length of the modulus *m* in octets, ***NULL*** is the empty string, and *kekLen* shall be set to the desired length of *KEK* (in octets).

Given *KEK*, a key-wrapping scheme WRAP and the symmetric key material *K* to be transported, the sender wraps *K* to get ciphertext C_2:

$$C_2 = \mathrm{WRAP}(KEK, K)$$

After this, the sender encrypts *Z* using the recipient's public RSA key *P* to yield C_1:

$$c_1 = \mathrm{RSA.ENCRYPT}(P,Z)$$

$$C_1 = \mathrm{I2OSP}(c_1, mLen)$$

Where RSA.ENCRYPT is the cryptographic primitive RSAEP in [PKCS-1] defined by

$$\mathrm{RSA.ENCRYPT}(P,Z) = Z^e \bmod m$$

The scheme output is $C = C_1 \mid C_2$ (C_1 concatenated with C_2) which is transmitted to the recipient. The decryption operation follows straightforwardly: the recipient recovers *Z* from C_1 using the recipient’s private key, converts *Z* to *KEK*, and then unwraps C_2 to recover *K*.

7.1.2 KDF

KDF is equivalent to the key derivation function KDF2 defined in [X9.44] (and KDF in [X9.42], [X9.63]). It is defined as a simple key derivation function based on a hash function. For the purposes of this specification, the hash function shall be SHA-1.

KDF takes three parameters: the shared secret value *Z*: an octet string of (essentially) arbitrary length, *otherInfo*: other information for key derivation, an octet string of (essentially) arbitrary length (may be the empty string), and *kLen*: intended length in octets of the keying material. *kLen* shall be an integer, at most $(2^{32} – 1)$hLen where *hLen* is the length of the hash function output in octets. The output from KDF is the key material *K*, an octet string of length *kLen*. The operation of KDF is as follows (note that "$\lceil n \rceil$" below denotes the smallest integer larger than, or equal to, *n*):

1) Let *T* be the empty string.

2) For *counter* from 1 to $\lceil kLen / hLen \rceil$, do the following:

Let D = 4-byte, unsigned big-endian representation of *counter*[1]

Let $T = T \mid \text{Hash}(Z \mid D \mid otherInfo)$.

3) Output the first *kLen* octets of *T* as the derived key *K*.

7.1.3 AES-WRAP

AES-WRAP is the symmetric-key wrapping scheme based on AES and defined in [AES-WRAP]. It takes as input a key-encryption key *KEK* and key material *K* to be wrapped. The scheme outputs the result *C* of the wrapping operation:

$$C = \text{AES-WRAP}(KEK, K)$$

7.2 Key Transport Mechanisms

7.2.1 Distributing K_{MAC} and K_{REK} under a Device Public Key

This section applies when protecting a Rights Object for a Device.

K_{MAC} and K_{REK} are each 128-bit long keys generated randomly by the sender. K_{REK} ("Rights Object Encryption Key") is the wrapping key for the content-encryption key K_{CEK} in Rights Objects. K_{MAC} is used for key confirmation of the message carrying K_{REK}.

The asymmetric encryption scheme RSAES-KEM-KWS shall be used with the AES-WRAP symmetric-key wrapping scheme to securely transmit K_{MAC} and K_{REK} to a recipient Device using the Device's RSA public key. An independent random value *Z* as described in section 7.1.1 shall be chosen for each encryption operation. For the AES-WRAP scheme, K_{MAC} and K_{REK} are concatenated to form *K*, i.e.:

$$KEK = \text{KDF}(\text{I2OSP}(Z, mLen), \textbf{\textit{NULL}}, kekLen)$$

$$C_2 = \text{AES-WRAP}(KEK, K_{MAC} \mid K_{REK})$$

$$C_1 = \text{I2OSP}(\text{RSA.ENCRYPT}(PubKey_{Device}, Z), mLen)$$

$$C = C_1 \mid C_2$$

where *kekLen* shall be set to 16 (128 bits) and *mLen* is the length of the modulus of the Device’s RSA public key in octets. In this way, AES-WRAP is used to wrap 256 bits of key data ($K_{MAC} \mid K_{REK}$) with a 128-bit key-encryption key (*KEK*).

After receiving *C*, the DRM Agent splits it into C_1 and C_2 and decrypts C_1 using its private key (consisting of a private exponent *d* and the modulus *m*), yielding *Z*:

$$C_1 \mid C_2 = C$$

$$c_1 = \text{OS2IP}(C_1, mLen)$$

$$Z = \text{RSA.DECRYPT}(PrivKey_{Device}, c_1) = c_1^{\,d} \bmod m$$

[1] Example: If *counter* = 946, *D* will be 00 00 03 b2

where OS2IP converts an octet string to a nonnegative integer and is defined in [PKCS-1].

Using Z, the Device can derive KEK, and from KEK unwrap C_2 to yield K_{MAC} and K_{REK}:

$$KEK = \mathrm{KDF}(\mathrm{I2OSP}(Z, mLen), \boldsymbol{NULL}, kekLen)$$

$$K_{MAC} \mid K_{REK} = \text{AES-UNWRAP}(KEK, C_2)$$

The following URI shall be used to identify this key transport scheme in **<xenc:EncryptionMethod>** elements:

http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-1#rsaes-kem-kdf2-kw-aes128

7.2.2 Distributing K_D and K_{MAC} under a Device Public Key

This section applies when provisioning a Device with a Domain key, K_D.

K_D is the symmetric key-wrapping key used when protecting K_{REK} and K_{MAC} in a Rights Object issued to a Domain D. K_D is a 128-bit long AES key generated randomly by the sender and shall be unique for each Domain D. K_{MAC} is used for key confirmation of the message carrying K_D.

In this case, exactly the same procedure as in the previous section shall be used, the only difference being the replacement of K_{REK} with K_D.

7.2.3 Distributing K_{MAC} and K_{REK} under a Domain Key K_D

This section applies when protecting a Rights Object for a Domain.

The key-wrapping scheme AES-WRAP SHALL be used. KEK in AES-WRAP SHALL be set to K_D and K to the concatenation of K_{MAC} and K_{REK}, i.e.:

$$C = \text{AES-WRAP}(K_D, K_{MAC} \mid K_{REK})$$

After receiving C, the DRM Agent decrypts C using K_D:

$$K_{MAC} \mid K_{REK} = \text{AES-UNWRAP}(K_D, C)$$

The following URI shall be used to identify this key transport scheme in **<xenc:EncryptionMethod>** elements:

http://www.w3.org/2001/04/xmlenc#kw-aes128

7.2.4 Distributing K_{MAC} and K_{MEK} under an RI Public Key

This section applies when protecting a Metering Report for an RI.

K_{MAC} and K_{MEK} are each 128-bit long keys generated randomly by the sender. K_{MEK} ("Metering Encryption Key") is the encryption key used to encrypt Metering Reports. K_{MEK} is a 128-bit long AES key generated randomly by the Device and shall be unique for each Metering Report. K_{MAC} is used for key confirmation of the message carrying K_{MEK}.

The asymmetric encryption scheme RSAES-KEM-KWS shall be used with the AES-WRAP symmetric-key wrapping scheme to securely transmit K_{MAC} and K_{MEK} to an RI using a RSA public key. An independent random value Z as described in section 7.1.1 shall be chosen for each encryption operation. For the AES-WRAP scheme, K_{MAC} and K_{MEK} are concatenated to form K, i.e.:

$$KEK = \mathrm{KDF}(\mathrm{I2OSP}(Z, mLen), \boldsymbol{NULL}, kekLen)$$

$$C_2 = \text{AES-WRAP}(KEK, K_{MAC} \mid K_{MEK})$$

$$C_1 = \mathrm{I2OSP}(\mathrm{RSA.ENCRYPT}(PubKey_{RI}, Z), mLen)$$

$$C = C_1 \mid C_2$$

where *kekLen* shall be set to 16 (128 bits) and *mLen* is the length of the modulus of the RIs RSA Public Key, in octets. In this way, AES-WRAP is used to wrap 256 bits of key data ($K_{MAC} \mid K_{MEK}$) with a 128-bit key-encryption key (*KEK*).

After receiving *C*, the recipient splits *C* it into C_1 and C_2 and decrypts C_1 using its private key (consisting of a private exponent *d* and the modulus *m*), yielding *Z*:

$$C_1 \mid C_2 = C$$

$$c_1 = \mathrm{OS2IP}(C_1, mLen)$$

$$Z = \mathrm{RSA.DECRYPT}(PrivKey_{RI}, c_1) = c_1^{\,d} \bmod m$$

where OS2IP converts an octet string to a nonnegative integer and is defined in [PKCS-1].

Using *Z*, the recipient can derive *KEK*, and from *KEK* unwrap C_2 to yield K_{MAC} and K_{MEK}.:

$$KEK = \mathrm{KDF}(\mathrm{I2OSP}(Z, mLen), \textbf{\textit{NULL}}, kekLen)$$

$$K_{MAC} \mid K_{MEK} = \mathrm{AES\text{-}UNWRAP}(KEK, C_2)$$

The following URI shall be used to identify this key transport scheme in **<xenc:EncryptionMethod>** elements:

http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-1#rsaes-kem-kdf2-kw-aes128

7.3 Use of Hash Chains for Domain Key Generation

To simplify Domain Key management when several generations of a Domain are expected (see section 8 for information on Domains), an RI may elect to make use of hash chains, and derive later Domain Keys from earlier ones. The procedure to do this is as follows: When creating the Domain, the RI generates a master Domain key, $\mathrm{K_M}$. The RI then applies KDF on $\mathrm{K_M}$ at least as many times n as the RI believes there will be generations of the Domain:

$\mathrm{DK_n} = \mathrm{K_M}$

$DK_{n-1} = \mathrm{KDF}(DK_n, \mathrm{NULL}, kekLen)$

$DK_{n-2} = \mathrm{KDF}(DK_{n-1}, \mathrm{NULL}, kekLen)$

…

$DK_0 = \mathrm{KDF}(DK_1, \mathrm{NULL}, kekLen)$

where *kekLen* = 16 and DK_j represents the Domain key with generation number j.

The result, DK_0, is then distributed as described in section 5.4.5.2 as the first key (generation number 0) for Domain *D*. When a Device in a Domain has been revoked, or the RI otherwise decides to create a new Domain generation (shift Domain key), the RI computes and distributes DK_1. Devices supporting this mechanism therefore only need to store DK_i, for the latest received Domain generation *i*, since for any earlier generation *j* ($j < i$), DK_j can be computed as:

$DK_{i-1} = \mathrm{KDF}(DK_i, \mathrm{NULL}, 16)$

$DK_{i-2} = \mathrm{KDF}(DK_{i-1}, \mathrm{NULL}, 16)$

… ($i – j$ applications of KDF)

DK_j = KDF(DK_{j+1}, NULL, 16)

RIs supporting this mechanism only need to store the current generation number i, the maximum number of generations n, and the Domain master key K_M.

Support for this mechanism is optional, both for RIs and Devices. As described in sections 5.4.5.1.1 and 5.4.5.2.1, the Device and RI negotiate the use of this mechanism during the 2-pass Domain join protocol.

8. Domains

8.1 Overview

A Domain is a set of Devices that possess a common Domain Key provisioned by a Rights Issuer. Devices in a Domain may share Domain Rights Objects and are able to consume and share any DCFs controlled by Domain Rights Objects.

The OMA DRM Domain concept is network centric. An RI defines the Domains, manages the Domain Keys, and controls which and how many Devices are included and excluded from the Domain. A user may request to add Devices to a Domain before acquiring Domain-bound content, or make these requests incrementally after receiving Domain-bound content.

A Domain is associated with a unique Domain Identifier, which includes a Domain Generation counter, and one or more Domain Keys. Multiple Domain Keys are a result of Domain upgrades performed by the Rights Issuer that manages the Domain. Each Domain Key corresponds to a specific Domain Generation. The value of the Domain Generation counter indicates the number of upgrades performed on the Domain.

Devices may join multiple Domains managed by one or more RIs.

8.2 Device Joins Domain

To join a Domain, a Device must have an RI Context established with the RI administering the Domain. A Device joining a Domain is the process of an RI authorizing a particular Device to be able to use all ROs for this Domain. When a Device joins a Domain it receives the necessary Domain information to be able to install Domain ROs.

A Device executes the Join Domain protocol (see 5.4.5) to join a given Domain. The result of a successful execution is the establishment in the Device of a Domain Context for the given Domain. The Domain Context includes Domain Key(s), Domain Identifier(s) and a Domain Expiry Time.

A Device MAY join multiple Domains managed by one or more RIs.

The Join Domain protocol is triggered by the **<joinDomain>** ROAP trigger.

If a Device joins a Domain with multiple Domain Generations (i.e. a Domain where more than one Domain Keys have been issued), the RI SHOULD issue to the Device the Domain Keys of all previous generations of the Domain, to allow use of all ROs bound to this Domain. But, if both the Device and RI are using the hash chain mechanism, the RI only needs to supply the most recent generation Domain key.

8.3 Domain RO Acquisition & Consumption

Domain ROs can be acquired by the same mechanism as Device ROs, using the 2-pass RO Request/Response protocol or the 1-pass RO Response protocol. The Device specifies the Domain Identifier in the RO Request. Domain ROs can also be acquired without being wrapped in a ROAP PDU, e.g. delivered to Devices as a result of a browsing session.

In order to consume a Domain RO, a Device MUST have a Domain Context for the Domain that the RO refers to. A Device MAY continue to consume Domain ROs that belong to a Domain where the Domain Context has expired. A Device SHALL NOT continue to consume Domain ROs that belong to a Domain where the Domain Consume Time has expired. See section 8.7.2.1 for the procedures for installing Domain ROs.

8.4 Device Leaves a Domain

In order for a Device to leave a Domain, it must assure the RI that it has deleted all information about the Domain that enables it to use any ROs for the Domain. When leaving a Domain a Device MUST delete the associated Domain Context, without a Domain Context ROs issued for that Domain will no longer be consumable. When leaving a Domain a Device

MAY, but is not required to, remove the corresponding Domain ROs and associated Content. The Device SHOULD obtain user confirmation before deleting Domain ROs and associated Content.

A Device MUST execute the Leave Domain protocol (see 5.4.5) to leave a Domain. A Device may do this by sending a LeaveDomainRequest message to the riURL as stored in the RI Context associated with the Domain Context or as a result of receiving a **<leaveDomain>** ROAP Trigger. The riURL from a <**leaveDomain**> ROAP trigger MUST be used if the LeaveDomain is triggered by a ROAP trigger (See section 5.1.12).

Prior to sending a Leave Domain Request, the Device MUST ensure that the corresponding Domain Context is deleted.

8.5 Domain Context Expiry

The **<notAfter>** element in the **roap:JoinDomainResponse** specifies the Domain context expiry. After the Domain Context has expired the DRM Agent MAY continue to consume existing Domain ROs (as per section 8.3). Installation of new Domain ROs is not permitted for an expired Domain without domain renewal (as per section 8.7.2.1).

8.6 Support for Multiple Domains per Rights Issuer

To provide flexibility in Domain management, the system supports multiple Domains per Rights Issuer. The Device SHALL support the ability to join multiple Domains for each RI Context it establishes.

To ensure that each DRM Agent is able to provide a minimum level of functionality, a Device SHALL support at least 6 Domains, distributed among the established RI Contexts in any proportion.

The Device MAY optionally support more than 6 Domains. These additional Domains may also be distributed among the established RI Contexts in any proportion.

8.7 Domain RO Processing Rules

8.7.1 Overview

As a general principle, the processing rules for inbound Domain ROs are agnostic to the origin of the Domain RO i.e. it does not matter whether the Domain RO was delivered OTA from an RI or copied from another Device. There is no binding to a specific transport mechanism or protocol.

Domain ROs MAY be delivered to the Device either in the course of the RO acquisition protocol, inside a DCF file, as a separate standalone MIME object, or as part of a MIME multipart/related message [RFC2387]. As part of the installation of an RO, the Device must perform integrity and authenticity checks and replay attack related checks as described below.

8.7.2 Inbound Domain RO

The Device MUST support receiving a Domain RO in a ROAP-ROResponse message.

The Device MUST support receiving a Domain RO as a separate object.

The Device MUST support receiving a Domain RO inside a DCF.

Before installing and using a Domain RO to render the media objects inside the associated DCF the Device MUST process the Domain RO as defined in chapter 8.7.2.1.

8.7.2.1 Installing a Domain RO

When a Device receives a Domain RO, it MUST determine if it has a valid RI Context with the RI that issued the RO, by comparing the value of the **roap:ROPayload**'s **<riID>** element with the RI Identifiers in all valid RI Contexts stored in the Device. If the value of the **<riID>** element does not match that of an RI Identifier in a valid RI Context, the Device SHALL

NOT install the Domain RO. In this case the Device MAY keep the Domain RO and MAY send an HTTP GET to the URL specified in the **riURL** attribute of the **roap:ROPayload**. An HTTP GET on this URL SHOULD return either a JoinDomain ROAP Trigger or a (X)HTML page that starts an interaction with the User which may eventually lead to a JoinDomain ROAP Trigger. It should be noted that in the event that a JoinDomain ROAP Trigger is returned and the Device does not have a valid RI context then the Device MUST automatically register with the RI (as specified in section 5.2.1) prior to sending a JoinDomainRequest message.

The Device MUST verify the signature of the Domain RO using the RI's Public Key. If the verification fails the Device SHALL NOT install the Domain RO. In this case the Device MAY request a new Rights Object by sending a HTTP GET to the RightsIssuerURL in the relevant DCF.

After the Device verifies the signature of the Domain RO, it MUST compare the **<domainID>** field within the Domain RO with the Domain identifiers for any valid Domain Contexts already established with the RI that issued the Domain RO, as identified by the **<riID>** field. There are three possible outcomes of this comparison:

1. The **<domainID>** field matches a Domain identifier in a valid Domain Context already established with the RI. The Device MAY install the Domain RO.

2. The Domain baseID of the **<domainID>** field matches the Domain baseID of a stored Domain identifier in a valid Domain Context already established with the RI, but the Domain Generation of the RO is greater than the Generation of the stored domain ID. The Device MAY attempt to upgrade the Domain by sending a ROAP-JoinDomainRequest to the riURL in the RI Context associated with the Domain Context. The Device may have to obtain user consent to contact the RI, section 5.1.13 defines when explicit user consent is required.

   If the Domain upgrade is successful, the Device MAY install the Domain RO. Otherwise the Device SHALL NOT install the Domain RO.

3. The Domain baseID of the **<domainID>** field does not match a Domain baseID in any valid Domain Context already established with the RI. The Device MAY attempt to join the Domain by sending an HTTP GET request to the URL specified in the **riURL** attribute of the **roap:ROPayload**. The Device may have to acquire the user's consent prior to sending the HTTP GET request, section 5.1.13 defines when explicit user consent is required.

At the point where the Device sends an HTTP GET request to the URL specified in the **riURL** attribute of the **roap:ROPayload** the RO installation process as specified within this section is effectively aborted, however, the installation process may be restarted as a result of subsequent user interaction, by some other Device specific means that is outside the scope of this specification or as a direct result of responding to a subsequent ROAP Trigger. As a result of an HTTP GET to this URL the RI can choose (using its own criteria) whether to allow the Device to join the Domain or not and SHOULD return either a JoinDomain ROAP Trigger or a (X)HTML page that starts an interaction with the User which may eventually lead to a JoinDomain ROAP Trigger. In the event that the RI chooses not to allow the Device to join the Domain the RI MAY offer the user the opportunity to acquire a Device RO.

Before installing a Domain RO, the Device MUST successfully verify the MAC (using the **<mac>** element of the **roap:ProtectedRO**). If this verification fails, the Device SHALL NOT install the Domain RO. In this case the Device MAY initiate the process of acquiring a new Rights Object as described in section 5.2.2.

If the Domain RO is stateful**,** then the Device MUST perform the replay protection related checks defined in Section 10.4.

If the Domain Context has expired (indicated by the Domain Context Expiry Time) the Device MUST NOT install ROs for this Domain.

In the case where the Domain RO is received within a DCF, if the Device cannot verify the signature of the Domain RO, the Device MAY leave the Domain RO as is within the DCF. The Device MAY request a valid RO for the DCF as described in section 5.2.2.

8.7.2.2 Postprocessing after installing the Domain RO

There are cases where a Device installs a Domain RO that it received separately from the DCF to which it refers. In these cases, the Device SHOULD insert a copy of the Domain RO into the corresponding DCF [DRMDCF-v2.1] as soon as possible after installation.

The Device MAY insert the Domain RO into the DCF at a later stage, for example when the user requests to render the DCF or send it out of the Device. The Device MAY insert more than one Domain RO into a single DCF, as long as all of the inserted RO's are valid and correspond to a Domain that it is a member of.

When the Device inserts a Domain RO into a DCF, it SHOULD remove from the DCF all Domain RO's corresponding to Domains that the Device is not a member of.

The Device SHOULD NOT insert a copy of the Domain RO into the corresponding DCF if it concludes, using an algorithm not defined in this specification, that sending the installed Domain RO to other Devices does not add value for the end user, for example if the Domain RO has expired.

If the Device finds multiple DCF instances bound to the installed Domain RO, it SHOULD insert a copy of the Domain RO into each one of them.

8.8 Domain Upgrade

A Rights Issuer may upgrade a Domain if, for example, a Domain Key has been compromised or if a Device in the Domain has been revoked. This will probably be a rare event, but may be necessary as a last resort to stop DRM Content from leaking out of the system in the clear.

In order to upgrade a Domain, an RI MUST change the Domain Key and MUST increment the Domain Generation by one. If the Domain Generation value reaches 999 the Domain becomes obsolete. An RI MUST NOT issue ROs for an obsolete Domain and MUST NOT allow new Devices to join an obsolete Domain.

A Domain upgrade does not result in any Domain Context being deleted in any Device. After an upgrade, Domain ROs issued before the upgrade may still be used and shared. This applies to all Devices (revoked and unrevoked) previously in the Domain, and to any new Devices added to the Domain after the upgrade.

A Rights Issuer performs a Domain upgrade using the Join Domain protocol (see sections 8.2 and 5.4.5). An RI MAY initiate this protocol for the purposes of Domain upgrade by sending a ROAP trigger to a Device whose Domain membership it wishes to upgrade. If a Device receives a Join Domain ROAP trigger, it SHOULD compare the **<domainID>** field with the domain ID for any Domains already established with the RI that sent the ROAP trigger, with the sending RI as identified by the **<riID>** field. There are two possible outcomes of this comparison:

1. The Domain baseID of the **<domainID>** field matches Domain baseID of a stored domain ID, but the value of the Domain Generation in the trigger is greater than the value stored by the Device. The incoming trigger represents a Domain upgrade, as described in this section. The Device SHOULD in this case silently upgrade the Domain using the Join Domain protocol.

2. If the Domain baseID of the **<domainID>** field does not match Domain baseID of a stored domain ID, then the Device is not a member of the Domain. The Device MUST execute the Join Domain protocol (see 5.4.5); just as if it was joining the Domain for the first time (see section 8.2).

8.8.1 Use of hash chains for Domain key management

To avoid storage of multiple keys per Domain in the Device and in the RI (for the purpose of using old and new Domain ROs after Domain upgrade) it is possible to have a relation between the Domain Keys using Hash Chains (see section 7.3), as illustrated in the example below. The Device MAY support Hash Chains and the RI MAY support Hash Chains.

Example 1. Without hash chains

When generating a new Domain, the RI generates:

- a unique Domain Identifier DI, the Domain Generation is set to 000.
- a random secret Domain Key DK_0

At Domain upgrade the Domain Generation g is increased by 1, which is reflected in the Domain Identifier, and a new Domain Key DK_g is generated. The old Domain Key(s) must be stored in RI and Device to allow use of ROs issued before the upgrade. When Devices join a Domain, all Domain Keys of this Domain are sent in the Protected Domain Info of ROAP-JoinDomainResponse (see ROAP protocol suite).

Example 2. With Hash Chains (optional)

When generating a new Domain, the RI

- generates a unique Domain Identifier DI, the Domain Generation is set to 000
- generates an initial master key K_M for the Domain
- selects the maximum number of generations *n* for this Domain (not larger than 999)
    - defines a sequence of Domain Keys using the method described in Section 7.3

Since old Domain Keys (with low generation value) are possible to efficiently derive from new Domain Keys (with higher generation value), it is only necessary to store the newest Domain Key in the Device (and corresponding Domain Identifier so the Domain Generation is known). For the RI it is sufficient to store DK_n (=K_M) and the current Domain Identifier.

9. Confirming RO Installation

All Connected Devices SHALL support RO install confirmation. Information about the status of RO installations can be used by the RI for purposes such as customer care or billing. The RI SHALL use the ROAP-ROResponse message to indicate that he wishes to receive installation information for each RO contained therein.

9.1 Sending ROConfirmRequest

When the DRM Agent receives a ROAP-ROResponse which requires confirmation, it SHALL attempt to install all ROs contained in the ROAP-ROResponse. It SHALL then return a ROAP-ROConfirmRequest message containing the installation status of all ROs to the relevant RI. The request should be sent to the ROAP URL in the original RO acquisition trigger.

If a Well-intentioned Attempt to send the ROAP-ROConfirmRequest is not successful, i.e. a valid ROAP-ROConfirmResponse is not received within a reasonable time, DRM Agent SHOULD continue to make Well-intentioned Attempts to send the message until a response is received.

9.2 Processing ROConfirmRequest

If the RI receives a ROAP-ROConfirmRequest message then the RI MUST respond with an appropriate ROAP-ROConfirmResponse message.

If the RI receives a ROAP-ROConfirmRequest message it MUST first check that it has a valid Device Context with the Device sending the message by checking the value of <**deviceID**> element of the ROAP-ROConfirmRequest message.

If the RI does not have a valid Device Context the RI MUST return a ROAP-ROConfirmResponse message with the value of the **<status>** element equal to *NotRegistered.*

If the device ID is valid, the RI must verify that the session ID in the ROAP-ROConfirmRequest message corresponds to the session ID in a previous ROAP-ROResponse delivered by this RI to the Device. If the session IDs do not match, the RI SHALL terminate the RO Confirmation installation protocol

If the RI has a valid Device Context and session ID it MUST validate the signature on the ROAP-ROConfirmRequest.

- If the RI cannot validate the Device signature on the ROAP-ROConfirmRequest message the RI MUST:
    - Return a ROAP-ROConfirmResponse with the status set to an appropriate value i.e. *'SignatureError',' NoCertificateChain', 'InvalidCertificateChain','TrustedRootCertificateNotPresent'.*

If the signature is valid, then the RI should verify that the RO IDs in the ROAP-ROConfirmRequest are valid. If any of the IDs are not valid, the RI MUST return a ROAP-ROConfirmResponse message with the value of the **<status>** element equal to *NotFound.*

9.3 Processing ROConfirmResponse

If the DRM Agent receives a ROAP-ROConfirmResponse message it MUST check the value of the **<nonce>** element, and the **"sessionID"** attribute. If this nonce value does not match the value of the **<nonce>** element sent in the preceding ROAP-ROConfirmRequest message or the session IDs do not match the Device MUST:

- Discard the ROAP-ROConfirmResponse,

Otherwise the Device MUST next check the signature of the ROAP-ROConfirmResponse message. If the signature is wrong, the Device MUST Discard the ROAP-ROConfirmResponse.

If the signature is correct, the Device MUST next check the value of the **<status>** element.

- If the value of the **<status>** element is equal to *'Success'* the DRM Agent can consider the RO confirmation protocol to have been successful.

- .If the value of the **<status>** element is not equal to '*Success'* the DRM Agent MUST
    - Follow the rules specified in section 5.3.6
    - Consider the attempt to send the ROAP-ROConfirmRequest as successful.

10.Protection of Content and Rights

10.1 Protection of Content Objects

The Content Objects are protected by symmetric key encryption. The details of the content format are specified in [DRMDCF-v2.1] document. Protecting content confidentiality is a key part of the DRM system. Only the intended Devices must be able to decrypt the content. To accomplish this content protection, the Rights Issuer MUST encapsulate the Content Encryption Key (CEK) in a Rights Object. This Rights Object, in turn, is protected as described in Section 7.2 to ensure that only the intended Devices may access the CEK and therefore the DRM Content.

For integrity protection of the DCF, a cryptographic hash value of the DCF SHOULD BE (if the Device is in possession of the DCF) generated and sent to the Rights Issuer for validation during RO acquisition (see Section 5.4.4.1.1). Also the DCF Hash MAY BE inserted into a Rights Object by the Rights Issuer. This hash value MUST BE generated according to the DCF hash calculation procedure specified in section 15.4. If the Rights Object contains a DCF hash value, DRM Agents in client Devices MUST verify once that this hash value is identical to the hash value calculated by the DRM Agent over the DCF. If the hash values are not identical, the DRM Agent MUST prohibit the DCF from being decrypted and used.

To summarize Devices are expected to calculate a DCF Hash:

- Once when initiating 2-pass RO Acquisition if the DCF is available on the Device; and
- Once before allowing access to the content for the first time after a new RO is received if the RO is delivered via another method than 2-pass RO Acquisition or if the DCF is not available on the Device at the time of sending the RO Request.

In a progressive download scenario, the DRM Agent can complete hash verification only after the complete DCF has been received and possibly after DCF decryption has started. The DRM Agent MUST discontinue DCF decryption and use, if the hash verification fails.

To improve user experience (by reducing waiting time due to verification of the DCF integrity), the DCF hash may be calculated by the Device in advance, possibly during download or as soon as the DCF is received, and may be cached by the Device for later use. In order to verify the integrity of the DCF a Device may compare the cached hash value to that in a corresponding RO. When acquiring an RO, the Device may also report the cached hash value to the RI in the RO-Request. The location for the cached hash value is not specified and therefore is implementation specific.

10.2 Composite Content Objects and Associated Rights Objects

10.2.1 Multiple Rights for Composite Objects

A Rights Object can contain one or more Permissions and Constraints (i.e. multiple rights). Each set of Permissions and Constraints is identified by a unique identifier, and uniquely associated with a Media Object by the identifier. One Rights Object may contain Permissions that are associated with Media Objects contained in separate DRM Containers (DCFs). Some example use cases include:

- Multiple DCFs delivered at different times (e.g. subscription-based MMS where several MMS messages are sent to a user)
- Multiple DCFs delivered at the same time but not encapsulated in a single package (e.g. streaming media (audio stream and video stream)).
- Multiple DCFs delivered at the same time and in a single package that is not a DCF (e.g. an MMS message containing several pictures, each encapsulated in its own DCF)

The Rights Objects can also specify permissions and constraints for each of the individual Media Objects within a Multipart DCF. In this case, the individual Media Objects can be referenced separately by the Rights Object associated with the Multipart DCF.

10.2.1.1 Multiple Rights for Multipart DCFs

A Multipart DCF contains multiple separate Media Objects, e.g., a theme consisting of a ringing tone and a logo. Each Media Object in a Multipart DCF is realised as a separate DCF Container with its own unique Content-ID. Every usage-permission, that is granted to the user in a Rights Object, refers to one or many of these DCF Containers. They are called "assets" in the REL specification, and contain the Content-ID of the correspondent DCF Container. It is important to note that permissions do not refer to entire DCFs but to DCF Containers.

An example shall clarify this: when a Multipart DCF contains an audio file and two images, a Content Provider can grant a user the permissions to play the audio data, display the two images, and print the second image three times. The corresponding Rights Object would contain three permission elements (<play>, <display>, and <print> along with the respective <constraint> elements), each of which would refer to the Multipart DCF Container it affects.

Note that if a Content Provider wants to set the same permissions for all Media Objects in a Multipart DCF, there are two possibilities to do so:

Each <permission> element contains a list of references (<asset> elements) to every Multipart DCF Container.

The <permission> element does not contain any <asset> elements. The REL specification states that in this case the permission elements <play>, <display> and <print> refer to all assets defined in the RO's <agreement> element (for details see [DRMREL-v2.1]).

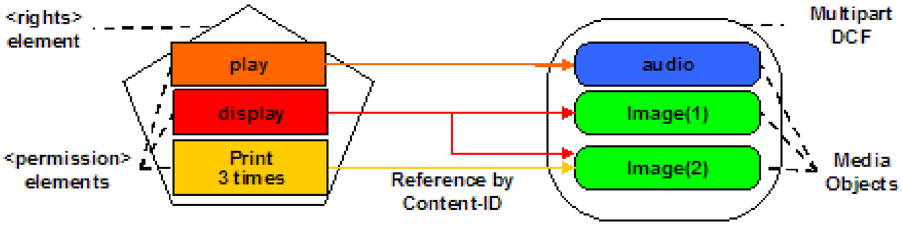

Figure 12: Multiple Rights for Multipart DCFs

Another case is where a Media Object is a Composite Object, i.e. it contains other Media Objects by means of inclusion. Such a Composite Object can have assigned only a single Content-ID which can be referenced by a Rights Object. Permission and Constraints expressed referring to the Composite Object MUST be applied to all individual Media Objects contained in the Composite Object (e.g. the images and audio files contained in a zip archive).

10.3 Protection of Rights Objects

In the OMA DRM Architecture, a given Content Object is associated with one or more Rights Objects. The Rights Object is made up of the required header information, security elements, and the rights information for the associated Content Object. The Rights Objects are acquired by the Device as a result of a successful completion of the Rights Object Acquisition Protocol or through sharing in a Domain.

Integrity protection prevents un-authorised modification of the rights information within the Rights Object. The syntax and semantics of the Rights Object is specified in the [DRMREL-v2.1] document, while this specification defines the use of [XML-DSIG] to create a digital signature over the set of elements that need integrity protection. The DRM Agent MUST verify the digital signature, when available, within the Rights Object, before the associated content is made available to the user. Use of the digital signature provides the client the ability to verify the authenticity & integrity of the information. The Rights Issuer MUST provide the certificate chain necessary to validate the signature either during the ROAP session or by use of "out-of-band" methods.

The Rights Object MUST be assigned a unique identifier by the Rights Issuer.

10.3.1 Device RO Processing Rules

10.3.1.1 Overview

A Device can acquire ROs through use of the Rights Object Acquisition protocol, or through their inclusion in DCFs. Before installation of a received RO, the Device must make a number of checks, including integrity, authenticity, and replay attack related checks as described below.

This section defines processing rules for Rights Objects specific to an individual Device. The processing rules for Domain ROs are found in section 8.7.

10.3.1.2 Receiving a Device RO

The Device MUST support receiving a Device RO in a ROAP-ROResponse message.

Before installing and using a RO to render any media objects inside the associated DCF(s), the Device MUST process the RO as defined in section 10.3.1.3.

10.3.1.3 Installing a Device RO

The Device MUST support receiving a Device RO in a ROAP-ROResponse message.

When a Device receives a Device RO through a successful execution of the RO Acquisition protocol, it MUST proceed as follows:

- Verifications:
    - If the Device RO was signed (i.e. the **<signature>** element is present in the **roap:ROPayload**), the Device MUST verify the signature using the RI's Public Key.
    - The Device MUST verify the MAC on the Device RO using the **<mac>** element of the **roap:ProtectedRO**.
    - The Device MUST verify that the **<riID>** element of the **roap:ROPayload** identifies the same RI as signed the **roap:ROResponse** message.
- The Device MUST inform the user and MUST NOT install the Device RO if any of the above verifications fail. Likewise, Device ROs received in unsuccessful executions of the RO Acquisition protocol MUST NOT be installed.
- If the RO is stateful (indicated by the **stateful** attribute of the **<ro>** element)**,** then the Device MUST perform the replay protection related checks defined in Section 10.4.

The Device MAY support receiving a Device RO in other ways than through a successful execution of the RO Acquisition protocol. In this case, the Device MUST proceed as follows:

- Verifications:
    - The Device MUST verify that the signature (i.e. the **<signature>** element in the **roap:ROPayload**) is present
    - The Device MUST verify the signature using the RI's Public Key.
    - The Device MUST verify the MAC on the Device RO using the **<mac>** element of the **roap:ProtectedRO**.

    - The Device MUST verify that the **<riID>** element of the **roap:ROPayload** matches the RI Identifier in any valid RI context

- The Device MUST inform the user and MUST NOT install the Device RO if any of the above verifications fail.

- If the Device RO is received within a DCF and if any of the above verifications fail the Device MAY leave the Device RO as is within the DCF. The Device MAY request a Rights Object for the DCF as described in section 5.2.2.

- If the **<riID>** element in the **roap:ROPayload** of a Device RO does not match the RI Identifier in any valid RI context the Device MAY send an HTTP GET to the URL specified in the **riURL** attribute of the **roap:ROPayload**. The Device may have to acquire the user's consent prior to sending the HTTP GET request, section 5.1.13 defines when explicit user consent is required. At the point where the Device sends an HTTP GET to the URL specified in the **riURL** attribute of the **roap:ROPayload** the RO installation process as specified within this section is effectively aborted, however, the installation process may be restarted as a result of subsequent user interaction, by some other Device specific means that is outside the scope of this specification or as a direct result of responding to a subsequent ROAP Trigger. An HTTP GET on the URL specified in the riURL attribute of the roap:ROPayload SHOULD return either a RegistrationRequest ROAP Trigger or a (X)HTML page that starts an interaction with the User which may eventually lead to a RegistrationRequest ROAP Trigger.

- If the RO is stateful (indicated by the **stateful** attribute of the **<ro>** element)**,** then the Device MUST perform the replay protection related checks defined in Section 10.4.

10.4 Replay Protection of Rights Objects

10.4.1 Introduction

Rights Objects containing permissions with constraint elements such as <count>, <interval>, or <accumulated> requires the current state of the usage permissions to be maintained in the DRM Agent. In contrast with stateless rights, there has to be a mechanism to protect against an attacker replaying the reception of such stateful ROs to the Device, which could cause an unauthorised extension of the permissions. But for those stateless ROs which have been uploaded to RI (see section 12), the Device MUST remove them and can't consume them any longer. So there also has to be a mechanism to protect against an attacker replaying the reception of such stateless ROs to the Device

In certain variants of RO acquisition described in this specification such a replay protection mechanism is inherent in the protocol. In particular, the 2-pass RO Acquisition protocol contains a Device nonce, sent in the RO request and sent back and signed in the RO response. The DRM Agent compares an incoming correctly signed RO Response with the nonce in a sent RO Request and unless there is a match, the RO is rejected and replay of the RO Response is not possible. RI authentication provided by the 2-pass protocol can thus be used to control replay.

In contrast, the 1-pass RO Acquisition protocol or the sharing of ROs in a Domain does not offer a challenge/response mechanism. 1-pass ROAP offers a limited replay protection through the time-based RI authentication, but it is not optimal in that the synchronisation between RI and Device cannot be guaranteed.

To accommodate for this, a local replay cache will be kept in the Device. Logically, the replay cache is a table where each entry contains a Globally Unique RO Identity (GUID) for a received, stateful RO, and the RI Time Stamp for the RO. The GUID MUST be unique for each instance of the RO (or else a user who legitimately twice in a row buys the same stateful RO could be seen as mounting a replay attack).

When stateful ROs with GUIDs and time stamps are received, they will be compared with previously received stateful ROs in the replay cache. If there is a match with an existing entry, the newly received RO will not be installed. When the replay cache is full, ROs with newer (later) time stamps replace entries with older time stamps and ROs with time stamps older than the oldest time stamp in the cache are rejected. This mechanism provides a secure replay protection. Appropriate sizing of the replay cache minimizes the risk that a long delivery time of one stateful RO in combination with mass distribution of other stateful ROs with later time stamps causes the delayed RO to be rejected (in situations of mass distribution of stateful ROs, the RI could use the 2-pass ROAP protocol since that has an inherent replay protection mechanism that does not interfere with the mechanism described here.).

A limitation of the method described above is that sharing of Domain ROs with very old time stamps may be affected by the finiteness of the replay cache. A second mechanism is therefore included to eliminate this limitation. This second mechanism defines a separate replay cache for ROs with a GUID, but without a timestamp. GUIDs of new ROs without timestamps will then be compared to GUIDs in the GUID-only replay cache. If there is a match, the RO is rejected; otherwise it is accepted and the replay cache is updated. If the GUID-only replay cache is full, a previous entry is removed to give room for the GUID of the new RO. This mechanism does not limit sharing of ROs but is possible to circumvent, since it is possible to replay stateful ROs with GUIDs that has been deleted from the cache.

The reason for having separate replay caches is that the secure mechanism based on timestamps and GUIDs should not be affected by the latter, more limited, replay protection mechanism. A separate replay cache for GUID-only entries still provides a certain degree of protection for corresponding ROs, allowing RIs to balance security interests against the risk of unintentional rejection of "old" Domain ROs.

To provide replay protection for stateless Rights Objects after they have been uploaded to an RI, Devices MUST maintain a separate local replay cache for stateless ROs. Logically, the replay cache is a table where each entry contains a Globally Unique RO Identity (GUID) for an uploaded stateless RO, and the RI Time Stamp for the RO. The GUID MUST be unique for each instance of the RO.

10.4.2 Replay Protection Mechanisms

This section defines two mechanisms enabling protection against Device RO as well as Domain RO replay attacks.

The OMA DRM Release 2 replay protection mechanisms are intended to support the use case of stateful Device ROs or Domain ROs that are delivered without a prior RO Request, i.e. in the 1-pass ROAP, or Domain ROs delivered outside of ROAP. In the case of Domain ROs, the statefulness is **per Device** in the Domain. E.g. if a Domain RO with a count 3 constraint is successfully shared between Devices, each Device is allowed 3 uses. It is the original Domain RO that SHALL be shared between Devices within a Domain. Any state information about how many times a constraint has been consumed, SHALL NOT be shared between the Devices.

The **roap:ROPayload** type contains two components for stateful RO replay protection management: the Globally Unique ID attribute **id** and the RI Time Stamp element **<timeStamp>**. In addition, the RI indicates that an RO is stateful by setting the **stateful** attribute to **True**. The **<timeStamp>** element is optional for Domain ROs and provides the RI with two different methods for replay protection: Replay protection with and without RI-assigned timestamps (RITS). These methods are described in the following.

A Device MUST have two (logical) replay caches: one with <GUID, RITS> entries for stateful ROs with GUIDs and RITS, and one with <GUID> entries for stateful ROs with GUIDs only. If the Device supports Rights Object Upload (see section 12), it MUST have an additional replay cache with <GUID, RITS> entries for such stateless ROs that have been uploaded to the RI. The Device MUST protect the integrity of its replay caches. It is RECOMMENDED that each replay cache for stateful RO is able to store at least 100 entries, but the size of the replay cache for stateless RO should be as large as is practical.

10.4.2.1 Stateful ROs with RI Time Stamps

This replay protection mechanism is applicable to both Device ROs and Domain ROs and is secure, i.e. it can guarantee protection against replay attacks. However, in the Domain case, subsequent sharing may be restricted by the replay protection mechanism and cannot be guaranteed. In particular, a receiving Device may reject Domain ROs that are shared long after they have been received from the RI. The mechanism assumes at least loosely synchronised time across the set of RIs and OCSP responders that may be accessed by a Device.

a) When receiving a stateful RO with a **<timestamp>** element (RITS), the Device MUST perform the following procedure: If the RITS is more than 24 hours in the future when compared to the Device's DRM Time then the Device MUST reject the RO. The user MUST be informed of the event and of the present Device DRM Time, and

SHOULD be asked if the Device’s DRM Time is correct. If the DRM Time is not correct the Device SHOULD initiate Device DRM Time synchronisation by re-registering with the RI using the Registration protocol.

b) Failing a), if the GUID for the RO is already in the <GUID, RITS> replay cache then the Device MUST reject the RO.

c) Failing b), if the <GUID, RITS> replay cache is not full, the Device MUST accept the RO and insert the ROs GUID and RITS values as an entry in the replay cache. Note: The GUID value is the **id** attribute of the **roap:ROPayload** value.

d) If the replay cache is full, and the RITS is before the earliest RI Time Stamp in the replay cache the Device MUST reject the RO.

e) Otherwise – if the replay cache is full, and the RITS is after the earliest RI Time Stamp in the replay cache the Device MUST accept the RO and insert the corresponding <GUID, RITS> values as an entry in the replay cache, by deleting the cache entry with the earliest RITS value.

10.4.2.2 Stateful ROs without RI Time Stamps

This replay protection mechanism is intended for Domain ROs. It does not restrict subsequent sharing, installation or usage of Domain ROs but it is less secure than the mechanism in Section 10.4.2.1 and it does not guarantee replay protection. Hence, if protection from replay of a stateful RO is important, the RI should include an RI Time Stamp in the RO payload. If indefinite sharing of stateful Domain ROs in a Domain is important and it is acceptable that, with some effort from an attacker, this stateful RO may be replayed, then the RI should not include an RI Time Stamp in the RO payload.

When receiving a stateful RO without a **<timestamp>** element, the Device MUST perform the following procedure:

a) If the RO's GUID is in the GUID-only replay cache then the Device MUST reject the RO.

b) Failing a), if the GUID-only replay cache is not full, the Device MUST accept the RO and insert the RO's GUID value as an entry in the cache.

c) Otherwise – if the GUID-only replay cache is full, the Device MUST accept the RO and insert the RO's GUID value as an entry in the GUID-only replay cache by deleting an existing entry in the cache. The Device MAY use FIFO in the GUID-only replay cache or MAY select a random entry for deletion.

10.4.2.3 Stateless ROs uploaded to RI

This replay protection mechanism is mainly intended for such stateless ROs that have been uploaded to RI. It can guarantee protection against replay attacks. As stated in section 5.3.10 all Device ROs contain a timestamp. This timestamp is used for replay protection.

When receiving a stateless RO with a **<timestamp>** element (RITS), the Device MUST perform the following procedure:

a) If the GUID for the RO is already in the <GUID, RITS> replay cache then the Device MUST reject the RO.

b) Failing a), if the <GUID, RITS> replay cache is not full, the Device MUST accept the RO

c) Failing b), if the replay cache is full, and the RITS is before the earliest RI Time Stamp in the replay cache the Device MUST reject the RO.

d) Otherwise – if the replay cache is full, and the RITS is after the earliest RI Time Stamp in the replay cache the Device MUST accept the RO.

When uploading a stateless RO to RI successfully, the Device MUST perform the following procedure:

a) If the replay cache is not full, the Device MUST insert the corresponding <GUID, RITS> values as an entry in the replay cache.

b) Otherwise – if the replay cache is full, the Device MUST insert the corresponding <GUID, RITS> values as an entry in the replay cache, by deleting the cache entry with the earliest RITS value. However, if the RITS of the RO to be inserted is before the earliest RITS value in the replay cache, there's no need to insert it.

10.5 Parent Rights Object

A Rights Object may inherit Permissions from another Rights Object, using the <inherit> syntax as specified in [DRMREL-v2.1]. This mechanism can be used, for example, to specify rights for content acquired as part of a subscription.

In this section, the Rights Object that inherits permissions is referred to as a Child Rights Object (C-RO). The Rights Object that contains the Permissions that are inherited is referred to as a Parent Rights Object (P-RO).

Client Devices MUST verify that the Child Rights Object and its related Parent Rights Object were issued by the same Rights Issuer before the associated content is made available to the user.

A Parent Rights Object MUST NOT contain any DCF hash value, as described in Section 10.1 since a Parent Rights Object does not reference any DRM Content directly.

10.5.1 Parent Rights Objects and Domains

A Rights Issuer MAY bind Child Rights Objects and Parent Rights Objects to a Device or to a Domain. The permission inheritance mechanism described in this section is independent of the cryptographic binding of the Rights Objects.

10.5.2 Semantics of stateful constraints

The DRM Agent MUST maintain the state of any stateful constraint relative to the Rights Object in which the constraint appears, and not relative to any single Media Object. If only one Media Object references a Rights Object, the DRM Agent will interpret the stateful constraints in the Rights Object as applying to that one Media Object. If more than one Media Object references a Rights Object, directly or by inheritance (see section 10.5 above), the DRM Agent MUST interpret the stateful constraints as applying collectively to all Media Objects that reference that Rights Object.

The figure below illustrates these semantics in the context of a Parent Rights Object. Two Child Rights Objects inherit permissions from a single Parent Rights Object. On the left side, each C-RO specifies a <play> permission with a count constraint. The constraint applies individually to the content item linked to the C-RO. The user may use DCF-1 up to 5 times, and DCF-2 twice. On the right side, the P-RO specifies the constrained <play> permissions. In this case, the user may use either DCF up to a total of 7 times. This may be 4 uses of DCF-1 and 3 uses of DCF-2, or even no uses of DCF-2 and 7 uses of DCF-1.

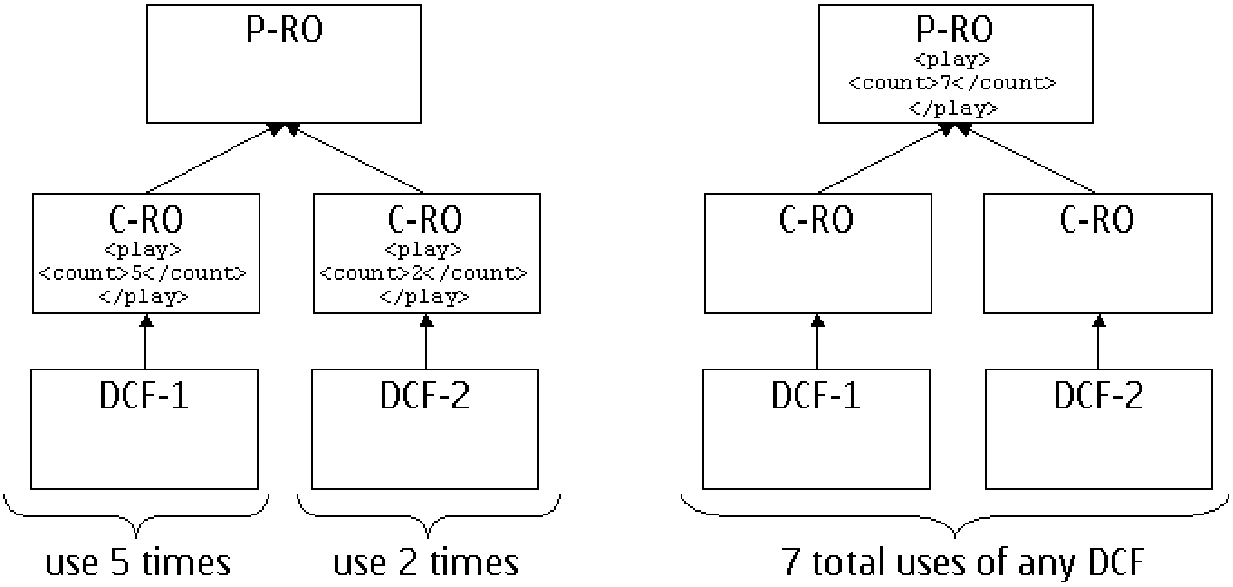

Figure 13: Parent ROs and Associated Semantics

10.5.3 Selection of Parent Rights Object

The issuing of a new Parent Rights Object may lead to a situation in which a DRM Agent has two valid Parent Rights Objects with the same value of the <uid> element of the <context> element of the <asset> element (see Appendix C.4 of [DRMREL-v2.1]).

At any given moment, a Child Rights Object is allowed to inherit permissions and constraints from only one single Parent Rights Object. Therefore, in case the DRM Agent has installed two or more valid Parent Rights Objects with the same value of the <uid> element of the <context> element of the <asset> element, and a permission is exercised from a Child Rights Object that refers to this <uid> value, the DRM Agent MUST select exactly one of these Parent Rights Objects from which the Child Rights Object is allowed to inherit permissions and constraints.

The selection of a Parent Rights Object is done according to the Rights Object evaluation order (see section 5.10 of [DRMREL-v2.1]).

10.6 Off-Device Storage of Content and Rights Objects

Because Devices have a limited amount of storage space in which to store DRM Content and Rights Objects, users may desire to move DRM Content and Rights Objects off the Device, e.g. to removable memory, a personal computer, or a network store to make room for new DRM Content and Rights Objects. A given Rights Object can be inserted into the corresponding DCF for purposes of storage and simplicity in managing the objects. At some later point in time, they may want to retrieve said DRM Content and Rights Objects from the remote storage back onto the Device store.

As explained in earlier sections of this specification, both the DRM Content and Rights Objects are protected and bound to a specific Device or a Domain. For this reason, DRM Content and Rights Objects MAY be allowed to leave the Device provided the following condition is met:

The DRM Content and the Rights Objects MUST be in a protected form, meaning they cannot be accessed by any other Device/Domain than the original intended Device/Domain to which the rights were issued.

10.7 Group ID Mechanism

A content object (DCF) MAY contain an OMADRMGroupID Box that defines the group identity of the DCF, as specified in [DRMDCF-v2.1].

For any DCF that contains a Group ID, a Rights Issuer MAY issue Rights bound to the DCF Content ID or bound to the DCF Group ID. The ROAP assumes that the client will present the correct RO identifier in the RO Info field of the RO Request message.

When a DRM Agent locates a Rights Object bound to the GroupID of any DCF in a group, it MUST obtain the DCF-specific key by decrypting the value of the GroupKey field in the DCF with the Content Encryption Key stored in the Rights Object.

When searching for a valid Rights Object for a DCF that includes an OMADRMGroupID Box, the DRM Agent may find Rights Objects bound both to the DCF GroupID and to the individual ContentID of the DCF. In this case the DRM Agent MUST process the Right Objects as specified in section 5.10 of [DRMREL-v2.1].

A Rights Object bound to a DCF Group ID MUST NOT include any DCF hash values, as described in Section 10.1. This allows a Rights Issuer to issue group Rights Objects to a client before delivery of content, and without specific knowledge of the group content that a client may acquire.

The Group ID mechanism described in this section is independent of the cryptographic binding of the Rights Object to a Device or a Domain. A Rights Issuer MAY bind a group Rights Object to a Device or to a Domain.

10.8 Semantics of Stateful Constraints

The DRM Agent MUST maintain the state of any stateful constraint relative to the Rights Object in which the constraint appears, and not relative to any single Media Object. If only one Media Object references a Rights Object, the DRM Agent will interpret the stateful constraints in the Rights Object as applying to that one Media Object. If more than one Media Object references a Rights Object, the DRM Agent MUST interpret the stateful constraints as applying collectively to all Media Objects that reference that Rights Object.

The figure below illustrates these semantics in the context of a Group Rights Object. Two DCFs refer a Group Rights Object which specifies the constrained <play> permissions. In this case, the user may use either DCF up to a total of 7 times. This may be 4 uses of DCF-1 and 3 uses of DCF-2, or even no uses of DCF-2 and 7 uses of DCF-1.

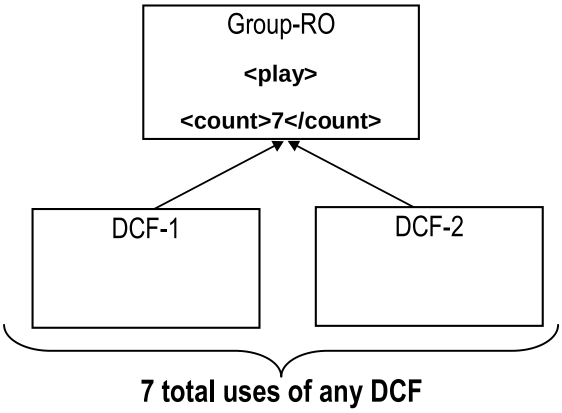

Figure 14: Group RO and Associated Semantics

11.Metering

Connected Devices MUST support Metering. Unconnected Devices that support DRM Time SHOULD support Metering. All normative statements apply to both Connected and Unconnected Devices that support DRM Time unless specifically stated otherwise. RIs MUST support Metering.

The OMA DRM metering solution can be summarised as follows:

- It allows an RI to define that Metering Information must be collected by the DRM agent when a particular RO is consumed through the inclusion of the <tracked> element in the RO as defined in [DRMREL-v2.1].

- It allows an RI to request the Device to report collected information through a Metering Report by sending a "Metering Report" ROAP Trigger.

Devices MAY have to obtain user consent before collecting Metering Information, section 5.1.13 defines when explicit user consent is required. User consent is obtained on a per-RI basis. If user consent has been given to collect Metering Information for a particular RI then it is considered that metering is enabled for all ROs issued by that RI. If user consent has not been given or if user consent has been retracted for collecting Metering information for a particular RI then metering is disabled.

When an RO contains the **<tracked>** element the DRM Agent must collect Metering Information if Metering is enabled. If Metering is not enabled, the DRM Agent does not collect Metering Information, however, it depends on the presence and the value of the *contentAccessGranted* attribute to determine whether or not the consumption of the associated DRM content is allowed. The detailed behaviour of the DRM Agent for Metering is described in Section 5.5.1 of [DRMREL-v2.1].

Upon reception of ROs that contain the **<tracked>** element the Device MUST store the value of the riID from the roap:ROPayload so that it can identify the RI that issued the RO.

11.1 Sending Metering Reports

The DRM Agent SHALL use the ROAP-MeteringReportSubmit message to send Metering Reports to the RI. This will be triggered if an RI sends a Metering Report ROAP Trigger to the Device.

If a DRM agent receives a MeteringReport ROAP Trigger from a given RI, and the DRM agent has collected Metering Information for this RI (that has not yet been reported) then DRM Agent SHALL send the ROAP-MeteringReportSubmit message. In the case that a User has previously consented to consume an RO that contains the <tracked> element from a particular RI, and Metering Information has been collected, but subsequently the user has retracted consent to collect further Metering Information for this RI, then the DRM Agent SHOULD send the Metering Report when a MeteringReport ROAP trigger is received. After user consent is denied for collection of Metering Information then only previously collected usage information is reported to the RI

If in the process of evaluating the RO for the purpose of consuming the associated DRM Content the DRM Agent determines that a permission has expired and if the **onExpiredURL** attribute is present within the corresponding permission the DRM Agent MUST send a HTTP GET request to the URL specified in the value of the **onExpiredURL** attribute at the first opportunity. A successful request to the URL MUST return a ROAP Trigger, a Download Descriptor or a bundled Download Descriptor and ROAP Trigger. If the value of the **onExpiredURL** attribute is a HTTP URL and the request fails with error code `404 Not Found` [RFC2616], the Device SHOULD NOT make further requests to the URL. If the request fails with some other error, the Device MAY retry the request at a later time.

It should be noted that the functionality associated with the **onExpiredURL** is not specific to Metering.

If at any point a Well-intentioned Attempt is not successful, the DRM Agent SHOULD continue to make Well-intentioned Attempts to send the Metering Report at the every available opportunity.

11.2 Processing of ROAP-MeteringReportSubmit Message

If the RI receives a ROAP-MeteringReportSubmit message then the RI MUST respond with an appropriate ROAP-MeteringReportResponse message.

If the RI receives a ROAP-MeteringReportSubmit message it MUST first check that it has a valid Device Context with the Device sending the Metering Report by checking the value of DeviceID element of the ROAP-MeteringReportSubmit message.

If the RI does not have a valid Device Context the RI MUST:

- Return a ROAP-MeteringReportResponse message with the value of the **<status>** element equal to *NotRegistered*.

If the RI has a valid Device Context it MUST:

- Check the Device Time as sent in the **<time>** element of the ROAP-MeteringReportSubmit message. If the RI does not consider this to accurate the RI MUST:
    - Return a ROAP-MeteringReportResponse with the status set to an appropriate value i.e. '*DeviceTimeError*.'
- If the RI considers the Devices DRM Time to be accurate the RI MUST Validate the signature on the MeteringReport.
    - If the RI can not validate the Device signature on the ROAP-MeteringReportSubmit message the RI MUST:
        - Return a ROAP-MeteringReportResponse with the status set to an appropriate value i.e. '*SignatureError*',' *NoCertificateChain*', '*InvalidCertificateChain*','*TrustedRootCertificateNotPresent*'.
    - If the RI can validate the Device signature the RI MUST validate the MAC on the Metering Report.
        - If the RI successfully validates the MAC the RI MUST decrypt the contents of the Metering Report as specified in section 11.4.2. In this case the value of the **<status>** element MUST be '*Success*'. If the RI can not decrypt the Metering Report the RI MUST return a ROAP-MeteringReportResponse with the value of the **<status>** element set to *UnableToDecryptMeteringReport*
        - If the RI is unable to validate the MAC on the Metering Report the RI MUST return a ROAP-MeteringReportResponse with the value of the **<status>** element set to *UnableToValdiateMeteringReportMAC*

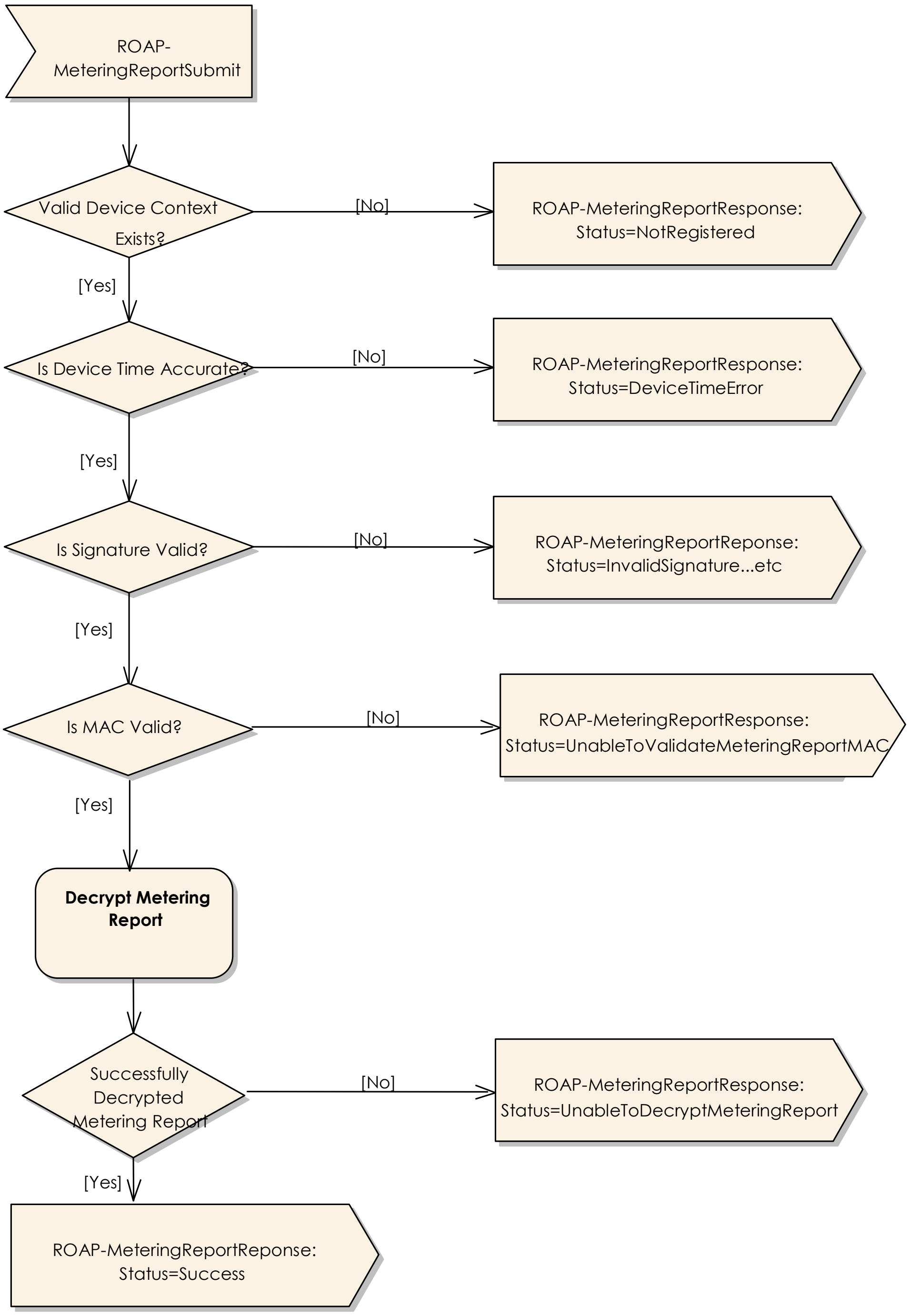
ROAP-MeteringReportSubmit
Valid Device Context Exists?
[No]
ROAP-MeteringReportResponse: Status=NotRegistered
[Yes]
Is Device Time Accurate?
[No]
ROAP-MeteringReportResponse: Status=DeviceTimeError
[Yes]
Is Signature Valid?
[No]
ROAP-MeteringReportReponse: Status=InvalidSignature...etc
[Yes]
Is MAC Valid?
[No]
ROAP-MeteringReportResponse: Status=UnableToValidateMeteringReportMAC
[Yes]
Decrypt Metering Report
Successfully Decrypted Metering Report
[No]
ROAP-MeteringReportResponse: Status=UnableToDecryptMeteringReport
[Yes]
ROAP-MeteringReportReponse: Status=Success

Figure 15: RI Processing of ROAP-MeteringReportSubmit Messages

11.3 Processing of ROAP-MeteringReportResponse Message

If DRM Agent receives a ROAP-MeteringReportResponse message it MUST check the value of the **<nonce>** element, if this nonce value does not match the value of the **<nonce>** element sent in the preceeding ROAP-MeteringReportSubmit message the Device MUST:

- Discard the ROAP-MeteringReportResponse and MUST continue to record Metering Information.

If the value matches the value of the **<nonce>** element sent in the preceeding ROAP-MeteringReportSubmit message the Device MUST verify signature of the ROAP-MeteringReportResponse message. If the verification of the signature was not successful the Device MUST:

- Discard the ROAP-MeteringReportResponse and MUST continue to record Metering Information.

If the signature verification was successful the Device MUST check the value of the **<status>** element.

- If the value of the **<status>** element is equal to '*Success'* the DRM Agent MUST
  - Delete any existing Metering Information that corresponds to the data sent in the Metering Report.
  - Consider the attempt to send the Metering Report as successful.
- If the value is equal to '*NoCertficateChain'* and the DRM Agent did not include its Certificate Chain in the corresponding ROAP-MeteringReportSubmit message the DRM Agent MUST resend the ROAP-MeteringReportSubmit message but include the appropriate Device Certificate Chain.
- If the value is not equal to '*Success'* or '*NoCertficateChain'*, the DRM Agent MUST
  - Follow the rules specified in section 5.3.6
  - Consider the attempt to send the Metering Report as successful.
  - NOT delete any existing Metering Information that corresponds to the data sent in the Metering Report and the DRM Agent MUST continue to record Metering Information.

If the signature verification was successful and the *Post Response URL* extension is present, the DRM Agent MUST send an HTTP GET request to the URL specified in the value of the **<prURL>** element of this extension at the first available opportunity. If the request fails with error code `404 Not Found` [RFC2616], the Device SHOULD NOT make further requests to the URL. If the request fails with some other error, the Device MAY retry the request at a later time.

If the signature verification was not successful, the DRM Agent SHOULD NOT send HTTP GET request to the URL given by the **<prURL>** element.

If the DRM Agent does not receive a ROAP-MeteringReportResponse message in response to sending a ROAP-MeteringReportSubmit message the DRM Agent MUST NOT:

  - consider the attempt to send the Metering Report as successful.
  - delete any existing Metering Information that corresponds to the data sent in the Metering Report and the DRM Agent MUST continue to record Metering Information until it receives a Metering Report ROAP Trigger.

The following figure shows this diagramtically.

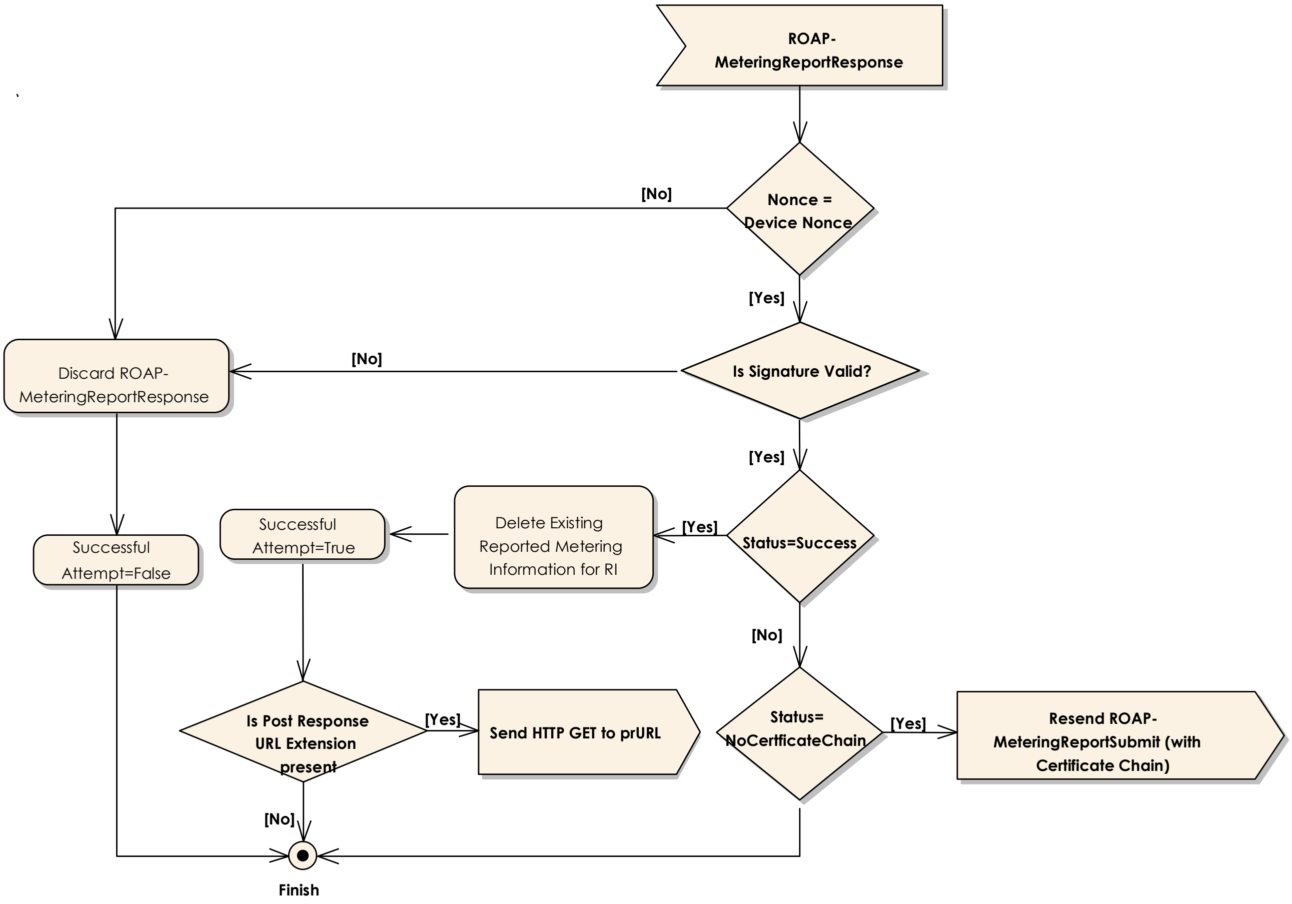
ROAP-MeteringReportResponse
Nonce = Device Nonce
[No]
[Yes]
Is Signature Valid?
[No]
Discard ROAP-MeteringReportResponse
[Yes]
Status=Success
[Yes]
Delete Existing Reported Metering Information for RI
Successful Attempt=True
Successful Attempt=False
[No]
Status= NoCertficateChain
[Yes]
Resend ROAP-MeteringReportSubmit (with Certificate Chain)
Is Post Response URL Extension present
[Yes]
Send HTTP GET to prURL
[No]
Finish

Figure 16: Processing of ROAP-MeteringReportResponse Messages

11.4 Metering Report Formatting

The Raw Metering Report MUST be formatted according to the ABNF syntax defined in this section. The ABNF notation used is defined in [RFC4234]. The terminals <DIGIT>, <WSP> and <CRLF> are also defined in [RFC4234].

```
rawMeteringReportData = *(CRLF perContentMeteringInformation) [CRLF]
perContentMeteringInformation= (contentID %d58 1*5(permission %d58 consumptionCount %d58
accumulatedConsumptionTime)
permission = play / display / execute / print / export

contentID = *(VCHAR)
consumptionCount=*(DIGIT)
accumulatedConsumptionTime=*(DIGIT) %d58 seconds
seconds=%x30-35 %x30-39
```

Where:

- contentID depends on the RO type used to consume the content :
    - if the content is authorised to be consumed by a group RO, contentID is the concatenation of :

        - the value of the **<UID>** element of the **<context>** element of an **<asset>** element and specifies the GroupId associated to the content
        - a separator (a semicolon)
        - the ContentID field in DCF's Common Header box and specifies the content identifier of the Metered Content to which the Metering Information applies
    - if the content is authorised to be consumed through a parent RO, contentID is the concatenation of :
        - the value of the <UID> element of the <context> element of an <asset> element and specifies the Parent RO ID associated to the content
        - a separator (a semicolon)
        - the ContentID field in DCF's Common Header box and specifies the content identifier of the Metered Content to which the Metering Information applies
    - Otherwise, contentID is the value of the **<UID>** element of the **<context>** element of an **<asset>** element and specifies the content identifier of the Metered Content to which the Metering Information applies.
- permission depends on the value of the <permission> element as defined in section §5.4.1 of [REL].
- AccumulatedConsumptionTime is the accumulated consumption time in minutes and seconds (separated by the ":") for a given permission
- consumptionCount is the number of consumption for a given permission

The DRM Agent MUST list for a dedicated content all the recorded metering information of all the permissions granted to this content.

In order to send the Metering Report to the RI the DRM Agent MUST insert the Raw Metering Report into the **<rawMeteringReportData>** element of an XML fragment of type **<meteringReportType>**.

11.4.1 Encryption of Metering Reports

To encrypt a Metering Report the DRM Agent MUST:

- Insert the value of a Raw Metering Report into the **<rawMeteringReportData>** element of a xml fragment of type **meteringReportType.**.
- Randomly generate a 128-bit long AES Metering Report Encryption Key, $K_{MEK.}$
- Randomly generate a 128-bit long AES Initialisation Vector.
- Randomly generate a 128-bit long MAC Key, K_{MAC}.
- Set the value of the **<EncryptionMethod>** of the **<EncryptedData>** element equal to:
    - http://www.w3c.org/2001/04/xmlenc#aes128-cbc
- Set the value of the **Type** attribute of the **<EncryptedData>** element equal to:
    - http://www.w3.org/2001/04/xmlenc#Content
- Use XML Encryption to encrypt the XML fragment of type **meteringReportType** using AES-128 in CBC mode and using the padding scheme defined in [RFC2630], the randomly generated Initialisation Vector should be prefixed to the encrypted data and then the combined IV and encrypted data should be inserted into the **<encryptedMeteringReport>** element of the ROAP-MeteringReportSubmit message.
- The Device MUST encrypt the Metering Encryption Key and MAC Key using the RI's Public Key as specified in section 7.2.4.
- Include the encrypted Metering Encryption Key and MAC key in the MeteringReportSubmit mesage as specified in section 5.4.6.1.2.
- Calculate a MAC on the canonical version according to Section 5.3.3 of the **<meteringReport>** element (excluding the **<mac>** element) using the MAC key $K_{MAC.}$ using MAC algorithm from the RI Context.
- Set the value of the **<mac>** element of the **<meteringReport>** element equal to this value.

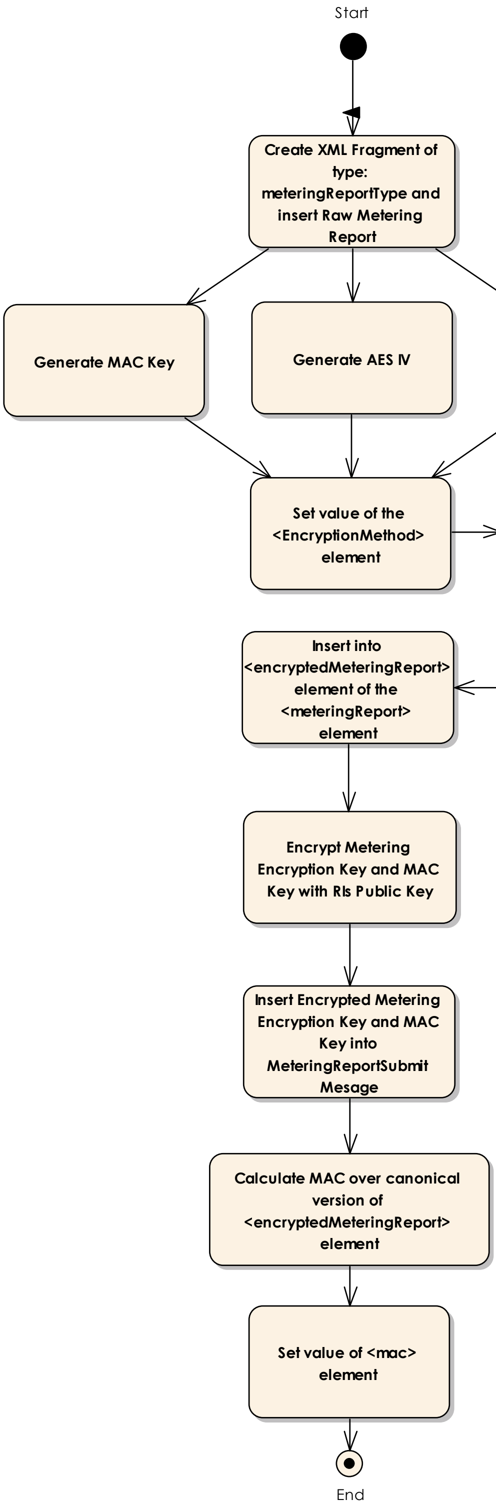
Start
Create XML Fragment of type: meteringReportType and insert Raw Metering Report
Generate MAC Key
Generate AES IV
Generate Metering Report Encryption Key
Set value of the <EncryptionMethod> element
Encrypt the XML Fragment
Prefix Encrypted Data with IV
Insert into <encryptedMeteringReport> element of the <meteringReport> element
Encrypt Metering Encryption Key and MAC Key with RIs Public Key
Insert Encrypted Metering Encryption Key and MAC Key into MeteringReportSubmit Mesage
Calculate MAC over canonical version of <encryptedMeteringReport> element
Set value of <mac> element
End

Figure 17: Encryption Process

11.4.2 Decryption of Metering Reports

In order to decrypt the contents of the **<meteringReport>** element of the **<encryptedMeteringReportData>** element of the ROAP-MeteringReportSubmit message the RI MUST:

- Check that the MeteringReportSubmit message is correctly formed, if it is not correctly formed the RI MUST respond with a ROAP-MeteringReportResponse with the value of the <status> element equal to 'MalformedMeteringReport'
- Unwrap the Metering Report Encryption Key, K_{MEK} and MAC Key, K_{MAC} (see section 7.2.4).
- Calculate a MAC on the canonical version according to Section 5.3.3 of the **<meteringReport>** element (excluding the **<mac>** element) using the MAC key $K_{MAC.}$ The MAC algorithm to use is defined in the Device Context.
- Check this value against the **<mac>** element of the **<meteringReport>** element of the ROAP-MeteringReportSubmit message.
    - If the calculated value is not equal to value of the **<mac>** element of the **<meteringReport>** element the RI MUST respond with a ROAP-MeteringReportResponse with the value of the **<status>** element equal to *'UnableToVerifyMeteringReportMAC'*
    - If the calculated value is equal to value of the **<mac>** element of the **<meteringReport>** element the RI MUST:
        - Extract the IV which is prefixed to the encrypted data.
        - Extract the Metering Report Encryption Key and use it to decrypt the contents of the **<EncryptedData>** element of the **<meteringReport>** element of the **<encryptedMeteringReport>** using the algorithm specified in the **<encryptionMethod>** element**.**
        - Decrypt the contents of the **<encryptedMeteringReport>** element; this should result in a XML fragment of type **meteringReportType**.
            - If this does not result in an XML fragment of type **meteringReportType** the RI MUST respond with a ROAP-MeteringReportResponse with the value of the **<status>** element equal to *'UnableToDecryptMeteringReport.'*

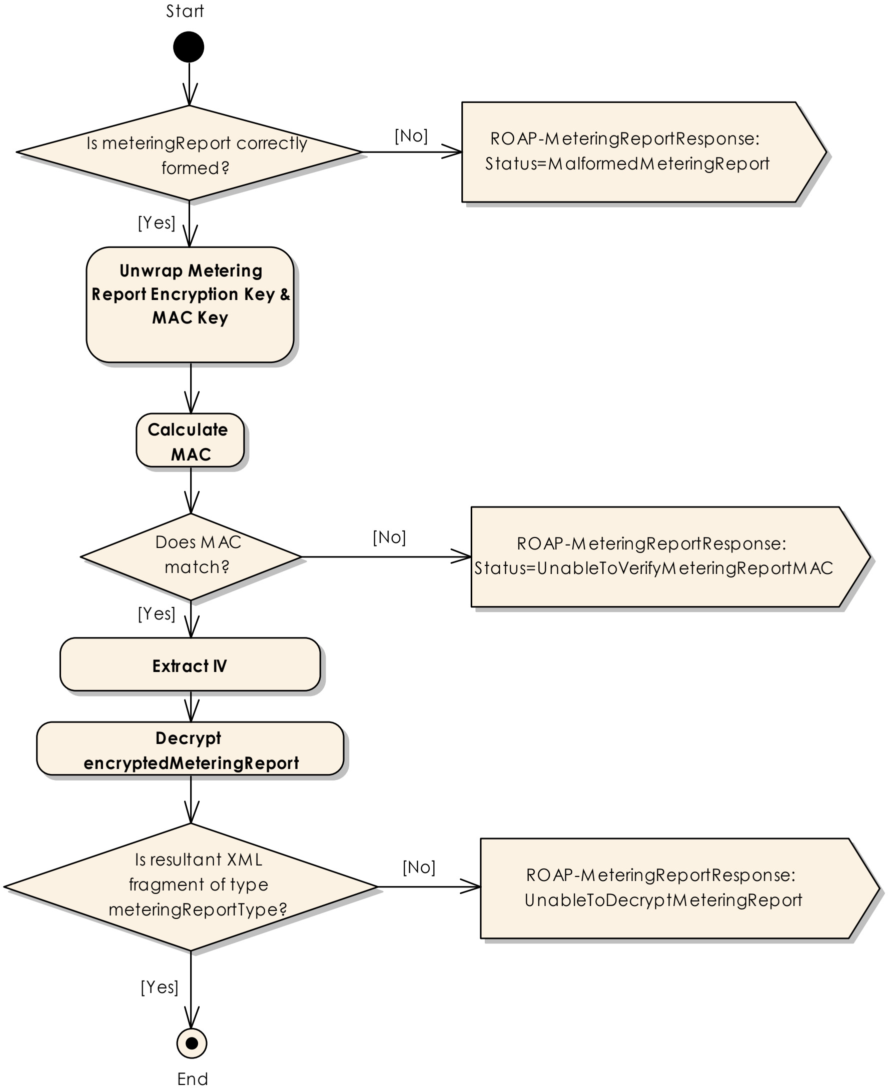
Start
Is meteringReport correctly formed?
[No]
ROAP-MeteringReportResponse: Status=MalformedMeteringReport
[Yes]
Unwrap Metering Report Encryption Key & MAC Key
Calculate MAC
Does MAC match?
[No]
ROAP-MeteringReportResponse: Status=UnableToVerifyMeteringReportMAC
[Yes]
Extract IV
Decrypt encryptedMeteringReport
Is resultant XML fragment of type meteringReportType?
[No]
ROAP-MeteringReportResponse: UnableToDecryptMeteringReport
[Yes]
End

Figure 18: Metering Report Decryption Process

11.5 Unconnected Device Support

By their definition Unconnected Devices have no wide area connection and rely on the “help” of Connected Devices in order to connect to RIs to send ROAP Messages. Connected Devices SHOULD support the functionality described in this section (this functionality is henceforth refered to as “Metering for Unconnected Devices” functionality) .

In terms of the functionality described within section 11 there is no reliable way for an RI to send an Unconnected Device a Metering Report ROAP Trigger, however, an Unconnected Device can attempt to initiate the sending a Metering Report (via a Connected Device). As specified in section [DRMREL-v2.1] a Device may initiate the sending of Metering Reports as a result of determining that a permission has expired. If the RI wishes the Device to attempt to send a Metering Report in this event the RI MUST include **onExpiredURL** attribute in a permission element of the RO.

When an Unconnected Device determines that a permission which contains the **onExpiredURL** attribute expires the Unconnected Device MUST ask the User to initiate a connection with the Unconnected Device from a Connected Device. In order to ensure that Users attempt this, the Unconnected Device MUST NOT allow any new ROs to be installed until it has made at least one Well Intentioned Attempt to send a request to the onExpiredURL.

A Connected Device that supports the Metering for Unconnected Device functionality MUST provide a suitable mechanism to allow the user to indicate that they want to establish a connection with an Unconnected Device. E.g. a suitable menu item. Once a connection with a Unconnected Device is available for this purpose the Connected Device MUST send the Unconnected Device a cdHello message. For this version of the specification the **v*ersion*** attribute of the cdHello message MUST be set to "**1.0**".

Upon receiving a cdHello message:

1. The Unconnected Device MUST Check the version attribute of the cdHello message. If the Unconnected Device does not support this version then it MUST respond with a ucdHello message with the status=*"VersionNotSupported"* and with the **version** attribute of the ucdHello message equal to the highest version supported by the Unconnected Device. If the Unconnected Device has a Metering Report to send then the Unconnected Device MUST include the **<httpURL>** element with the value equal to value of the **onExpiredURL** attribute from the expired permission.

2. If the Unconnected Device does not have any pending Metering Reports to send the Unconnected Device MUST respond with an ucdHello message with the status equal to *"NothingToSend"*. Upon receiving this status the Connected Device MUST close the connection and MAY inform the User.

3. If the Unconnected Device has an outstanding attempt to send Well-intentioned Attempt to an onExpiredURL then the Unconnected Device MUST respond with a ucdHello message with the status equal to *"Success"* and with the **<httpURL>** element present and the value equal to value of the **onExpiredURL** attribute from the expired permission.

Upon reception of a ucdHello message which contains a **<httpURL>** element a Connected Device MUST:

- Maintain the connection with the Unconnected Device and

- Attempt to send a HTTP GET Request to the URL in the ucdHello message. The Connected Device MUST append the following HTTP header to the HTTP GET Request:

  X-Oma-Drm-ucdcdRequest: "X-Oma-Drm-ucdcdRequest"

Upon reception of a HTTP GET request to a URL that was defined in the ***onExpiredURL*** attribute of an permission that it issued, the RI MUST Respond with either:

- An HTTP response with the status equal 200 OK and with the HTTP body containing either:
  - a ROAP-Trigger;
  - a Download Descriptor that points to a ROAP Trigger;
  - a ROAP Trigger bundled with a Download Descriptor.;
- Or a HTTP response with status not equal to 200 OK:
  - In this case the Connected Device MUST send a cdURLError message to the Unconnected Device and include the numerical part of the HTTP status code in the httpStatusCode attribute.

If an Unconnected Device receives a cdURLError message after sending a ucdHello message and the httpStatusCode attribute is equal to `404 (i.e. 404 Not Found` [RFC2616]), the Unconnected Device MUST not make any further requests to the URL that was sent in the ucdHello message and can consider the attempt to send the Metering Report as successful. If the request fails with some other error, the Unconnected Device MUST NOT treat the attempt to send the Metering Report as successful and MAY make further attempts to send a Metering Report to this URL.

If a ROAP Trigger is returned the ***proxy*** attribute MUST be present and MUST be set to 'True'.

Upon reception of this ROAP Trigger the Connected Device MUST behave as described in sections 14.6 and 18 and forward this to the Unconnected Device.

The cdHello, ucdHello and cdURLError messages are a simple XML messages, the schema fragment below defines these messages.

```
<simpleType name="ucdHelloStatus">
    <restriction base="string">
        <enumeration value="Success"/>
        <enumeration value="NothingToSend"/>
        <enumeration value="VersionNotSupported"/>
    </restriction>
</simpleType>

<element name="cdHello">
    <complexType>
        <attribute name="version" type="roap:Version" use="required"/>
    </complexType>
</element>

<element name="ucdHello">
    <complexType>
        <sequence>
            <element name="httpURL" type="anyURI" minOccurs="0"/>
        </sequence>
        <attribute name="status" type="roap:ucdHelloStatus" use="required"/>
        <attribute name="version" type="roap:Version" use="required"/>
    </complexType>
</element>

<element name="cdURLError">
    <complexType>
        <attribute name="version" type="roap:Version" use="required"/>
        <attribute name="httpStatusCode" type="positiveInteger" use="required"/>
    </complexType>
</element>
```

The **version** attribute is a <major.minor> representation of the ucdHello and cdHello messages that the Device supports. For this version of the specification, **version** SHALL be set to "**1.0**". Minor version upgrades must always be backwards compatible.

The **status** attribute is returned in an ucdHello message and indicates the status of the request. Valid values are as follows:

Success: The Unconnected Device successfully received the cdHello message and has responded appropriately

Nothing To Send: The Unconnected Device does not have a pending Metering Report to Send

VersionNotSupported: The Unconnected Device does not support the version of the cdHello message sent.

The <**httpURL**> element MUST contain a URL equal to the value specified in the **onExpiredURL** attribute (as defined in [OMAREL-v2.1]) of the expired permission which triggered the sending of a Metering Report.

The file extension for the cdHello, ucdHello and cdURLError messages MUST be “.ouc”.

12. Uploading Rights Objects

The Rights Object Upload (RO Upload) capability is optional for both Devices and Rights Issuers Issuers and applicable only to Device Rights Objects and not to Domain Rights Objects. During an RO Upload the Device sends to the RI information about one or more Device Rights Objects (each identified by their ROID) including the current state information (for Stateful Rights Objects). After successfully uploading an RO, the RI is able to re-issue the RO to a new Device belonging to the same User. How the RI determines that a new Device belongs to the same User as an old Device is out of scope of this specification. If a Device supports RO Upload, it MUST provide a means for the User to initiate the RO Upload. For example the Device MAY offer a menu option that enables the User to select one or more Rights Objects to be uploaded to an RI. If an RI supports RO Upload, it MUST record all issued Device ROs and SHOULD record the corresponding Device IDs for which the ROs were issued.

12.1 Sending ROUploadRequest

In the case of a Device initiated RO Upload the DRM Agent SHALL initiate the ROAP-ROUploadRequest to the riURL in the RI Context when the user wants to upload RO(s) to the RI.

The RO Upload can be initiated by a ROAP roUpload trigger. In this case the ROAP-ROUploadRequest MUST be sent to the <roapURL> in the trigger as described in section 5.2.1. In case where ROUploadRequest is initiated by the roUpload trigger, the DRM Agent MUST include all valid ROs which were issued from the RI, in the ROUploadRequest message.

Before sending the ROAP-ROUploadRequest, the DRM Agent MUST disable the RO(s) to be uploaded. If the RO Upload was initiated by a trigger, the DRM Agent MUST ask for user consent, before disabling the Rights.

The DRM Agent MUST send the current state information of the RO(s) to be uploaded to the RI if any of the ROs are stateful.

12.2 Processing ROUploadRequest

This section is applicable only to Rights Issuers that support RO Upload. If a Rights Issuer that does not support RO Upload receives a ROAP-ROUploadRequest message, it SHALL return a ROAP-ROUploadResponse with the **status** attribute set to "*NotSupported*".

If the RI receives a ROAP-ROUploadRequest message it MUST first check that it has a valid Device Context with the Device sending the message by checking the value of <**deviceID**> element of the ROAP-ROUploadRequest message.

If the RI does not have a valid Device Context the RI MUST return a ROAP-ROUploadResponse message with the **status** attribute set to "*NotRegistered".*

If the Device ID is valid, the RI MUST validate the signature on the ROAP-ROUploadRequest message. If the signature does not match, the RI MUST return a ROAP-ROUploadResponse message with the **status** attribute set to an appropriate value i.e. '*SignatureError',' NoCertificateChain', 'InvalidCertificateChain','TrustedRootCertificateNotPresent'*

If all the verifications above are valid, then the RI SHOULD verify that the ROID to be uploaded was issued by itself and that the corresponding RO is a Device RO. If the RI determines that the ROID to be uploaded was not issued by itself , it MUST return a ROAP-ROUploadResponse message with the **status** attribute set to "*UnknownUploadedRO*". If the RI determines that the ROID to be uploaded was issued by itself, but that the corresponding RO is not a Device RO or is expired, the RI MUST return a ROAP-ROUploadResponse message with the **status** attribute set to "InvalidUploadedRO".

Upon successful verification by the RI that the ROID to be uploaded was issued by itself and that the RO is a Device RO which is not expired, the RI SHOULD verify that the RO was issued to the Device that signed the ROAP-ROUploadRequest message. If upon such checking, the RI determines that the RO was issued to a different Device, the RI MUST reject the ROUploadRequest as invalid and MUST return a ROAP-ROUploadResponse message with the **status** attribute set to "*UnknownUploadedRO*".

If all the verifications are valid, then the RI SHOULD save the current state information for the uploaded RO.

12.3 Processing ROUploadResponse

If the DRM Agent receives a ROAP-ROUploadResponse message it MUST check the value of the **<nonce>** element. If this nonce value does not match the value of the **<nonce>** element sent in the preceding ROAP-ROUploadRequest message, the Device MUST discard the ROAP-ROUploadResponse.

Otherwise the Device MUST next check the signature of the ROAP-ROUploadResponse message. If the signature is wrong, the Device MUST discard the ROAP-ROUploadResponse.

If the signature is correct, the DRM Agent MUST next check the value of the **<status>** element.

- If the value of the **<status>** element is '*Success'* the DRM Agent can consider the RO(s) have been successfully uploaded and MUST remove the local RO(s) which have been successfully uploaded to the RI. In order to avoid replay attack as specified in section 10.4, the Device should keep track of the stateless RO's uploading history as specified in section 10.4.2.3.

- If the value of the **<status>** element is '*UnknownUploadedRO' or 'InvalidUploadedRO'* the DRM Agent MUST enable the RO(s) whose ROID has been indicated in the <**extension**> element of the ROAP-ROUploadResponse.

- If the value of the <**status**> element is any other value the DRM MUST reenable the ROs which were attempted to be uploaded.

In the case that no valid ROAP-ROUploadResponse has been received, the Device MAY resend exactly the same ROAP-ROUploadRequest. Additionally, until a valid ROAP-ROUploadResponse has been received, the Device MAY allow the user to resend exactly the same ROAP-ROUploadRequest. The Device SHALL NOT generate a fresh ROAP-ROUploadRequest containing ROs that are still disabled due to an earlier ROAP-ROUploadRequest. How often and when the ROAP-ROUploadRequest is resent is left to the implementation.

The RI MUST ensure that each ROAP-ROUploadRequest message is processed only once. Therefore, it MUST cache identifications of received ROAP-ROUploadRequest messages (e.g. device ID and message nonce) for a certain period of time, defined by the trust model. This period of time is called the *cache period*. The cache period is an interval moving over time, with its upper bound being the current time, and its lower bound a time in the past, at a fixed distance from the current time.

- The RI SHALL discard ROAP-ROUploadRequest messages which have been created before the cache period. This avoids the RI having to maintain an infinite cache, whereas old ROAP-ROUploadRequest messages cannot be used for a replay attack.
- If the RI receives an already processed ROAP-ROUploadRequest which has been created during the cache period it MUST answer with a ROAP-ROUploadResponse. It is RECOMMENDED that the RI sends the same status in the ROAP-ROUploadResponse as it sent after the receipt of the original RO. If the RI sends another status than it originally sent, the ROAP-ROUploadResponse is such that duplication of rights is not possible: if the RI sent a 'Success' status in the original ROAP-ROUploadResponse, the RI SHALL NOT send any other status than 'Success' in the new ROAP-ROUploadResponse, even if the newly received message is invalid (e.g. has a false signature).

12.4 Restoring uploaded RO

When the user wants to restore the ROs, the RI SHOULD verify first that the user of the restoring Device is the same user with the uploading Device. The method of user verification is beyond the scope of this specification. After this user verification is completed successfully, the RI can send ROAP-ROAcquisition trigger to the restoring Device. Then the RI can issue new RO(s) to the restoring Device according to the information of the ROID and state saved. The state information saved will be used as the original state information for the new RO(s). After issuing the new RO(s) successfully, the RI SHOULD delete the uploaded RO info, and record only the information for the newly issued RO.

13.Capability Signaling

13.1 Overview

When Devices contact Content Issuers and Rights Issuers, the Devices need to advertise their capabilities. This allows Content Issuers and Rights Issuers to customize content, purchase options, and so forth based upon the features and functionalities of the Device, thereby improving the overall user experience. OMA DRM relies upon two mechanisms for advertising Device capabilities: HTTP headers [HTTP] and User Agent Profile [UAProf].

13.2 HTTP Headers

When a Device uses HTTP to communicate with Content Issuers and Rights Issuers, the Device MUST advertise support for the following media types using the HTTP Accept header:

application/vnd.oma.drm.ro+xml	(DRM Rights Object)
application/vnd.oma.drm.dcf	(DRM Content Format)
application/vnd.oma.drm.roap-pdu+xml	(DRM ROAP PDUs)
application/vnd.oma.drm.roap-trigger+xml	(DRM ROAP Trigger)
application/vnd.oma.drm.roap-trigger+wbxml	(DRM ROAP Trigger – WBXML format)

In addition to the supported media types, Devices MUST advertise the DRM version using the “<major>.<minor>” format defined as BNF according to [RFC2234] below. The version number advertised by OMA DRM v2 Devices MUST match the DRM Enabler Release version that the Device supports.

```
DRM Version = “DRM-Version” “:” 1*DIGIT “.” 1*DIGIT
```

13.3 User Agent Profile

OMA DRM v2 Devices SHOULD advertise supported DRM methods, permissions, constraints, media types,, version and if supported, its external DRM capabilities using [UAProf]. "External DRM" refers to a DRM system to which the Device is able to export OMA DRM Content, for example, a DRM system used on a memory card. See Appendix H.6 for an example.

If the Device supports [UAProf], then the Device MUST advertise the attributes in the table below as indicated in the “MUST Advertise” column.

The attributes pertaining to an external DRM system MUST be included if the Device is capable of exporting OMA DRM Content to such a system. The attributes MUST NOT be included if the Device is incapable thereof.

UAProf Attribute	Description	Example Values	MUST Advertise
DrmClass	DRM v1 Conformance Classes as defined in [DRM]	"ForwardLock", "CombinedDelivery", "SeparateDelivery"	“ForwardLock” plus other supported DRM v1 methods
DrmPermissions	Optional DRM permissions that are supported as defined in [DRMREL] or **Error! Reference source not found.**.	“play”, “display”, “execute”, “print”	Supported permissions using the same syntax as defined in the REL specification.
DrmConstraint	Optional DRM permission	"datetime", "interval",	Supported constraints using

	constraints as defined in [DRMREL] or [DRMREL-v2.1]	“accumulated”	the same syntax as defined in the REL specification.
DrmMediaTypes	Media types the Device supports inside a DCF	"image/gif", "audio/midi", "video/3gpp"	Media types supported inside a DCF, expressed as MIME media types [RFC2045].
DrmVersion	DRM Enabler Release version supported by the client	“2.0”	Supported DRM Enabler Release version in “<major>.<minor>” format.
ExtDrmName	Name of the external DRM	“urn:oma:drms:oma-drms:secure-card-v1”	A textual name of the external DRM system expressed as a URN. Names for external DRM systems are managed by OMNA.
DrmMetering	Indicates if the DRM Agent supports Metering.	“True”, “False”	“True” if the DRM Agent supports Metering.

Table 20: User Agent Profile Attributes

13.4 Issuer Responsibilities

When a Content Issuer or Rights Issuer receives a request from a Device indicating that the Device supports OMA DRM version 2.x (any minor version of the DRM v2 enabler), the:

- Content Issuer MAY issue DRM v1.0 Forward Locked Content.
- Content Issuer MAY issue DRM v1.0 Combined Delivery Content and the Rights Object within the DRM Message is DRM v1.0 Rights Object, only if the Device advertises support for Combined Delivery.
- Content Issuer MAY issue DRM v1.0 Separate Delivery Content only if the Device advertises support for Separate Delivery.
- Content Issuer MAY issue DRM v1.0 Separate Delivery Content encapsulated in a DRM Message, only if the Device advertises support for Separate Delivery.
- Rights Issuer MAY issue a DRM v1.0 Rights Object for DRM v1.0 Separate Delivery Content, only if the client advertises support for Separate Delivery.
- Content Issuer SHOULD issue DRM v2.1 content.
- Content Issuer MAY issue DRM v2.x Content encapsulated in a DRM Message.
- Rights Issuer SHOULD send the ROAP Trigger to initiate the ROAP protocol (see section 5.2.1).
- Rights Issuer SHOULD NOT send the **<extendedTrigger>** ROAP Trigger to DRM 2.0 Devices.
- Rights Issuer MUST NOT issue any DRM v2.x Rights Object for DRM v1.0 Combined Delivery Content, even if the Device advertises support for Combined Delivery.
- Rights Issuer MUST NOT issue any DRM v2.x Rights Object for DRM v1.0 Separate Delivery Content, even if the Device advertises support for Separate Delivery.
- Rights Issuer MUST NOT issue any DRM v1.0 Rights Object for DRM v2.x Content.

- When issuing Rights Objects for DRM 2.x (P)DCFs:
    - If issuing a Domain bound Rights Object, the Rights Issuers SHOULD issue DRM 2.1 Rights Objects.
    - If issuing a Device bound Rights Object to a DRM 2.0 Device, the Rights Issuer SHOULD issue a DRM 2.0 Rights Object.
    - If issuing a Device bound Rights Object, to a DRM 2.1 Device the Rights Issuer SHOULD issue a DRM 2.1 Rights Object.

NOTE: When issuing DRM 2.1 Domain Rights Objects, the Rights Issuer SHOULD consider how that Rights Object will be handled by DRM 2.0 Devices, as is discussed in [DRMREL-v2.0].

14.Transport Mappings

14.1 Introduction

The following sections describe how ROAP PDUs are delivered using typical delivery protocols, the most common being HTTP. Examples are provided in Appendix H.1.

A Connected Device MUST support HTTP for transporting ROAP PDUs as described in section 14.2.

Connected Devices MAY support other ROAP transport mappings. Additionally Connected Devices MAY support the functionality to provide connectivity for an Unconnected Device as described in section 18.

An Unconnected Device SHALL support the functionality to use connectivity provided by a Connected Device, as described in section 18.

14.2 HTTP Transport Mapping

The following sections describe how ROAP PDUs are delivered using typical delivery protocols, the most common being HTTP. Examples are illustrated in Appendix H.2.

14.2.1 General

Connected Devices and Rights Issuers MUST support ROAP over HTTP, and HTTP MUST be supported according to [HTTP]. Note that the HTTP transport mapping also applies to [WSP].

14.2.2 HTTP Headers

Connected Devices and Rights Issuers MUST support the HTTP *Content-Type* header. This header describes the media type that is present in the body part of the HTTP *Request/Response*.

The DRM Agent MUST include an HTTP *Accept* header when sending a ROAP request over HTTP. The *Accept* header specifies the media types the DRM Agent will accept in response to the request.

Implementations MAY support other HTTP headers than those specified herein. The presence of HTTP headers other than those specified here when a ROAP message is received over HTTP SHOULD NOT by itself cause termination of the ROAP session.

14.2.3 ROAP Requests

- The DRM Agent MUST send ROAP request PDUs as the body of HTTP POST requests. Example:

```
POST /ro.cgi?roID=qw683hgew7d  HTTP/1.1
Host: ri.example.com

Content-Type: application/vnd.oma.drm.roap-pdu+xml

... [ROAP PDU] ...
```

In the above example the DRM Agent is using the *Request-URI* field for specifying the path component. The absolute URI of the Rights Issuer is specified using the HTTP *Host* header.

- The DRM Agent SHOULD use persistent connections when sending ROAP requests over HTTP.

- The DRM Agent SHALL indicate to the RI that the message is a ROAP PDU using the HTTP *Content-Type* header with value *application/vnd.oma.drm.roap-pdu+xml*. The following is an example of such a header field:

Content-Type: application/vnd.oma.drm.roap-pdu+xml

- The DRM Agent SHALL use the HTTP *Accept* header to indicate acceptable media types in response to ROAP requests sent over HTTP. The DRM Agent MUST accept at least the following media types:

  - application/vnd.oma.drm.roap-pdu+xml

  - multipart/related

  Example:

  Accept: application/vnd.oma.drm.roap-pdu+xml, multipart/related

- HTTP requests from the DRM Agent MUST NOT contain more than one ROAP request message.

14.2.4 ROAP Responses

- The Rights issuer MUST send ROAP response PDUs as the body of HTTP responses.

- The HTTP *Content-Type* header MUST be set to *application/vnd.oma.drm.roap-pdu+xml* when a ROAP PDU constitutes the message-body of a response. Example:

  Content-Type: application/vnd.oma.drm.roap-pdu+xml

- The Rights Issuer MAY send a ROAP response PDU as one part of a multipart message-body. In this case, the HTTP *Content-Type* header MUST be set to *multipart/related* (and include boundary information), and the body-part containing the ROAP response PDU MUST have a *Content-Type* header set to *application/vnd.oma.drm.roap-pdu+xml*. The following is an example of the former header:

  Content-Type: multipart/related; boundary=”XX---XX”

- In case the HTTP *Content-Type* header value in the ROAP Response does not match any of the *Accept* types in the corresponding ROAP Request, the Device SHALL terminate the ROAP session.

- The Rights Issuer MUST NOT include more than one ROAP response PDU in an HTTP response.

- The Rights Issuer MUST include an HTTP *Cache-Control* header with the value *no-transform* when sending an integrity-protected ROAP PDU. The *no-transform* directive prohibits network caches from doing any content transformations. The following is an example of the same:

  Cache-Control: no-transform

14.2.5 HTTP Response Codes

A Rights Issuer that refuses to perform a ROAP message exchange with a DRM Agent SHOULD return a 403 (Forbidden) response. In the case of an HTTP error while processing an HTTP request, the Rights Issuer MUST return a 500 (Internal Server Error) response. This type of error SHOULD be returned for HTTP-related errors detected before control is passed to the ROAP engine, or when the ROAP engine reports an internal error (for example, the ROAP schema cannot be located). If the type of a ROAP request cannot be determined, the Rights Issuer MUST return a 400 (Bad request) response code.

In these cases (i.e. when the HTTP response code is 4xx or 5xx), the content of the HTTP body is not significant.

In all other cases, the Rights Issuer MUST respond with 200 (OK) and a suitable ROAP message (possibly with ROAP-related error information) in the HTTP body.

DRM Agents MUST be able to handle HTTP response codes specified here (200, 400, 403, and 500).

14.3 OMA Download OTA

A Rights Issuer MAY use OMA Download OTA 1.0 [DLOTA] when delivering Content and Rights Objects in order to take advantage of managed download features such as terminal capability negotiation and installation notification. For example, a Rights Issuer may use the OMA Download OTA 1.0 installation notification as a billing trigger. The method described in this section MUST NOT be used for OMA Download OTA 2.0. For OMA Download OTA 2.0 the `license` attribute shall be used to deliver Rights in combination with Content as outlined in [DLOTA20].

Depending on specific deployment and business scenarios, OMA Download OTA can be used in different ways in the context of delivering DRM Content and Rights Objects. Appendix H.3 illustrates this with the help of a few examples, but is not meant to be an exhaustive collection of deployment scenarios.

14.3.1 Download Agent and DRM Agent Interaction

The Download Agent must collaborate with the DRM Agent when OMA Download OTA is used to deliver content and/or Rights Objects. In general, the DRM Agent will participate in the “Installation” phase of the Download OTA protocol.

The Download OTA protocol utilizes a Download Descriptor (DD) to provide information to the user and the Device prior to initiating the content object download. The following sections describe how the Download Agent and DRM Agent should behave when the Download Descriptor is used for DRM purposes.

14.3.1.1 Downloading DRM Content

When using Download OTA to download a DRM protected content object (that is, an encrypted content object packaged in the DRM content format), the Download Descriptor:

- MUST include *type* attribute indicating the MIME type for each of the object(s) to be downloaded.
- MUST include a *size* attribute indicating the size of the entire DRM Content Format object (which includes the encrypted content object)
- MUST include an *objectURI* attribute pointing to the protected content object.
- MAY include other optional attributes.

The Download Agent will process the Download Descriptor and perform the content object download as defined in [DLOTA].

The Download Agent SHOULD support the *nextURL* attribute of the Download Descriptor since two or more download transactions may be concatenated by the *nextURL*. For instance, the *nextURL* attribute of the Download Descriptor of a first download transaction for a DCF may point to the Download Descriptor of a second download transaction for the download of a ROAP Trigger.

14.3.1.2 Downloading ROAP Trigger or Rights Objects

A Device supporting OMA Download OTA for the download of a ROAP Trigger or a Rights Object, MUST support the co-delivery method. For the co-delivery method, as defined in the [DLOTA] the Download Descriptor MUST be the first entity in the multipart, and the ROAP Trigger, or the Rights Object MUST be the second entity of the multipart.

If Download OTA separate delivery method is used, the *objectUR*I in the Download Descriptor will provide the URL for retrieving the ROAP Trigger. The Download Agent, upon receiving the Download Descriptor, will retrieve the ROAP Trigger and deliver it to the DRM Agent for installation as defined in the Download OTA specification.

When using OMA Download OTA for delivery of the ROAP Trigger or the Rights Object, the Download Descriptor:

- MUST include *type* attribute(s) indicating the MIME type(s) of the object(s) being downloaded.

- MUST include an *objectURI* attribute containing the Content-ID of the ROAP Trigger in the multipart if the ROAP Trigger is co-delivered with the Download Descriptor. Otherwise, this attribute MUST contain the URL with which the Device may retrieve the ROAP Trigger

- MUST include a *size* attribute indicating the size of the object(s) being downloaded. If the Rights Object is co-delivered with the Download Descriptor, the size SHALL be calculated to be the size of the Protected ROs in the ROAP RO Response PDU as defined in section 5.4.4.2. Otherwise, this attribute MUST contain the size of the ROAP Trigger.

- MAY include other optional attributes.

When the Download Agent receives the ROAP Trigger (either via co-delivery or separate delivery), the Download Agent MUST send the ROAP Trigger to the DRM Agent after processing the Download Descriptor as defined in [DLOTA]. [DLOTA] requires that the Download Agent should use the information in the Download Descriptor to give the user a chance to confirm that they want to install the media object. In order to provide the desired end user experience, there is one exception:

> Where the value of a *type* attribute in the Download Descriptor indicates the MIME type of the ROAP Trigger or of the Rights Object then the Download Agent SHOULD NOT present the user with a user confirmation.

Upon receiving the ROAP Trigger, the DRM Agent MUST initiate the ROAP as defined in section 5.1.9 . The DRM Agent MUST notify the Download Agent of installation success or failure (including an appropriate error code as defined in [DLOTA]).

As defined in OMA Download OTA, the Download Agent MUST make a best effort attempt to send an installation status report to the Rights Issuer provided the *installNotifyURI* is present in the DD.

In case the download OTA is used for downloading the Rights Object using 1-pass delivery without the ROAP Trigger itself then the Rights Object (instead of the ROAP Trigger) is co-delivered with the Download Descriptor.

14.3.1.3 Downloading DRM Content and Rights Object Together

When using OMA Download OTA to download a Rights Object, the Download Descriptor MUST be co-delivered with the ROAP Trigger to download DRM Content and Rights Object together. If OMA Download OTA is used to download the Rights Object and DRM Content in a single multipart message, the Download Descriptor:

- MUST include *type* attribute(s) indicating the MIME type(s) of the object(s) being downloaded.

- MUST include an *objectURI* attribute containing the Content-ID of the ROAP Trigger in the multipart.

- MUST include a *size* attribute indicating the size of the Rights Object plus the size of the DRM Content. The size of the Rights Object SHOULD be calculated to be the size of the Protected ROs in the ROAP RO Response PDU as defined in section 5.4.4.2. If the size of the Protected ROs is not known at the time when the Download Descriptor is created, the server MAY use an estimate of the expected size of the Protected ROs.

- MUST include one or more type attributes with the content type of the protected content objects

- MAY include other optional attributes.

When the Download Agent receives a Download Descriptor and the ROAP Trigger, the Download Agent MUST send the ROAP Trigger to the DRM Agent after processing the Download Descriptor as defined in [DLOTA]. Upon receiving the ROAP Trigger, the DRM Agent MUST initiate the ROAP as defined in section 5.1.11. The Rights Issuer MUST provide the RO Response PDU and DRM Content in a multipart/related media type [RFC2387]. The RO Response PDU MUST be the first entry in the multipart and the DRM Content MUST be the second entry in the multipart. The DRM Agent MUST extract the RO Response PDU and DRM Content from the multipart and process both entities. The DRM Agent MUST notify the Download Agent of installation success or failure. As defined in [DLOTA], the Download Agent MUST make a best effort attempt to send this installation status to the Rights Issuer provided the installNotifyURI is present in the DD.

14.4 WAP Push

14.4.1 Push Application ID

The well-known value for the Push Application ID of the DRM User Agent Push remains unchanged from OMA DRM 1.0:

- URN: x-wap-application:drm.ua
- Number: 0x08

Rights Issuers and Content Issuers MUST use this Push Application ID when using WAP Push to deliver DRM Content or Rights Objects to the DRM Agent.

14.4.2 Content Push

A ROAP-ROResponse PDU MAY be delivered using WAP Push [PUSHOTA].

The DRM Agent MUST be able to receive and process a ROAP PDU that is pushed using the Push Application ID defined above.

A ROAP Trigger MAY be delivered using WAP Push [PUSHOTA].

The DRM Agent MUST be able to receive and process ROAP Triggers that are pushed to the DRM Agent using the Push Application ID defined above.

14.5 MMS

The Multimedia Messaging Service (MMS) may be used to transfer DRM Content in MMS Protocol Data Units (PDUs) as defined in [MMSENC]. In regard to DRM two use cases need to be distinguished:

a) The DRM Content is referenced from within the SMIL presentation description of the multimedia message (i.e. it can be rendered as part of a multimedia presentation).

b) The DRM Content is not referenced from within the SMIL presentation description of the multimedia message. This content can be handled by the Device independently from MMS.

If a multipart DCF is referenced from within the SMIL presentation description the first object is assumed to be the referenced Content Object. The reference for the Content Object is the header field “Content-Location” or “Content-ID” which is associated with the body part of the MMS message. This must not be confused with the Content-ID inside the DCF, which is used as a reference for the Rights Object.

An Example is provided in Appendix H.4.

14.6 ROAP over OBEX

14.6.1 Overview

OBEX is a protocol for exchanging objects. It can be used with Bluetooth, infrared, USB and RS232 and other bearers. The requirements of this document refer to [OBEX] version 1.3.

OBEX is a session-oriented protocol, which allows multiple request/response exchanges in one session. An OBEX session is initiated by an OBEX CONNECT request, and is established when the other Device returns a success response. The connection is terminated by sending a OBEX DISCONNECT request.

In this specification a Connected Device that supports the functionality to provide connectivity for Unconnected Devices (as specified in section 18) MUST contain an OBEX client and an Unconnected Device MUST contain an OBEX server.

When a session has been established, ROAP messages originating from the RI MUST be transferred from the Connected Device to the Unconnected Device using the OBEX PUT method. The Unconnected Device acknowledges the data, by sending a response with a status code, and possibly also containing some ROAP data.

ROAP requires that an OBEX connection is established. Connectionless OBEX cannot be used with ROAP.

Example messages are provided in Appendix H.5.

14.6.2 OBEX Server Identification

The ROAP-OBEX server is identified by the following UUID (to be used as a value for the "Target" header in OBEX CONNECT operations):

1d29f667-3236-374a-bf63-3c7a023bb17d

14.6.3 OBEX Profile

14.6.3.1 OBEX operations

The table below shows the OBEX operations that are used by the ROAP OBEX profile. Connected Devices that support the functionality to provide connectivity for Unconnected Devices (as specified in section 18) and Unconnected Devices MUST support these OBEX operations.

OBEX Operation	Opcode
Connect	0x80
Disconnect	0x81
Put	0x02 (0x82)
Get	0x83
Abort	0xFF

14.6.3.2 OBEX headers

The table below shows the OBEX headers that are used in the ROAP OBEX profile. Connected Devices that support the functionality to provide connectivity for Unconnected Devices (as specified in section 18) and Unconnected Devices MUST support these OBEX headers.

OBEX Header	Header Identifier	Comment
Type	0x42	application/vnd.oma.roap-pdu+xml
Length	0xC3	
Target	0x46	Required in CONNECT requests.
Who	0x4A	Identifies responding server in responses to CONNECT requests
Connection Id	0xCB	Value is set by the Unconnected Device in response to the CONNECT operation
Body	0x48	Carries ROAP PDUs; present if there is a need to send the PDU in several chunks.
End of Body	0x49	Carries ROAP PDUs.

14.6.3.3 OBEX Connect

The OBEX CONNECT operation SHALL contain the following OBEX fields and headers:

Field/Header	Name	Explanation/Value
Field	Opcode for CONNECT	0x80
Field	Packet length	Varies
Field	OBEX version	0x10 (for version 1.0 of the OBEX *protocol*)
Field	Flags	Varies; normally all zero
Field	Maximum packet length	Varies
Header	Target	1d29f667-3236-374a-bf63-3c7a023bb17d

The response code to a successful OBEX CONNECT operation SHALL be 0xA0. The following fields and headers SHALL be present in the response:

Field/Header	Name	Explanation/Value
Field	Response code	0xA0 for success
Field	Packet length	Varies
Field	OBEX version	0x10 (for version 1.0 of the OBEX *protocol*)
Field	Flags	Varies; normally all zero
Field	Maximum packet length	Varies
Header	Who	Shall have same value as the preceding "Target" header
Header	Connection ID	Identifies the connection

14.6.3.4 OBEX Disconnect

An OBEX DISCONNECT request SHALL contain the following fields and headers:

Field/Header	Name	Explanation/Value
Field	Opcode for DISCONNECT	0x81
Field	Packet length	Varies
Header	Connection ID	As established in the response to the CONNECT operation

The response code to a successful OBEX DISCONNECT operation SHALL be 0xA0. The following fields and headers SHALL be present in the response:

Field/Header	Name	Explanation/Value
Field	Response code	0xA0 for success
Field	Packet length	Varies

14.6.3.5 OBEX Abort

Note: The OBEX ABORT operation MAY be used to abort a multi-packet operation before it would normally end.

The OBEX ABORT operation SHALL, when requested, contain the following fields and headers:

Field/Header	Name	Explanation/Value
Field	Opcode for ABORT	0xFF
Field	Packet length	Varies
Header	Connection ID	As established in the response to the CONNECT operation

The response code to a successful OBEX ABORT operation SHALL be 0xA0 (or else the client will simply disconnect the OBEX connection). The following fields and headers SHALL be present in the response:

Field/Header	Name	Explanation/Value

Field/Header	Name	Explanation/Value
Field	Response code	0xA0 for success; otherwise the client will disconnect with a failure indication
Field	Packet length	Varies

14.6.3.6 OBEX PUT

The OBEX PUT operation SHALL contain the following OBEX fields and headers:

Field/Header	Name	Explanation/Value
Field	Opcode for PUT	0x02 or 0x82 (0x02 is used for non-terminal chunked messages, 0x82 is used for the terminal packet in a chunked message)
Field	Packet length	Varies
Header	Connection ID	Varies
Header	Type	application/vnd.oma.roap-pdu+xml or application/vnd.oma.roap-trigger+xml
Header	Body, End of Body	End of Body identifies the last chunk of an object; for other chunks the Body header shall be used.

In addition to these headers, the Length header MAY be used to indicate the complete length of an object The response code to a successful OBEX PUT operation SHALL be 0xA0 or 0x90, depending on whether the PUT operation was non-final (0x02) or final (0x82). The following fields and headers SHALL be present in the response:

Field/Header	Name	Explanation/Value
Field	Response code	0x90 (Continue) or 0xA0 (Success) for success
Field	Packet length	Varies

In addition, the following headers SHALL be present when the result of the OBEX PUT operation triggers the transmission of a ROAP message from the unconnected Device to the ROAP server (through the Connected Device):

Field/Header	Name	Explanation/Value
Header	Type	application/vnd.oma.roap-pdu+xml
Header	Body, End of Body	End of Body is used for last chunk of an object, for other chunks the Body header shall be used.

The response code shall be 0x90 when the size of the ROAP message in the response requires "chunking" (see [OBEX]). In this case, and in order to retrieve remaining parts, the Connected Device shall issue OBEX GET requests until it receives a response with response code 0xA0 (see below).

14.6.3.7 OBEX GET

The OBEX GET operation SHALL contain the following OBEX fields and headers:

Field/Header	Name	Explanation/Value
Field	Opcode for GET	0x83
Field	Packet length	Varies
Header	Connection ID	Varies
Header	Type	application/vnd.oma.roap-pdu+xml

The response code to a successful OBEX GET operation SHALL be 0xA0 or 0x90, depending on whether the message contains the complete (final part) of the object or not. The following fields and headers SHALL be present in the response:

Field/Header	Name	Explanation/Value
Field	Response code	0x90 (Continue) or 0xA0 (Success) for success
Field	Packet length	Varies
Header	Body, End of Body	End of Body is used for last chunk of an object, for

Field/Header	Name	Explanation/Value
		other chunks the Body header shall be used.

The response code shall be 0x90 when the size of the object requires "chunking." In this case, and in order to retrieve remaining data, the Connected Device SHALL continue to issue OBEX GET requests until it receives a response with response code 0xA0.

14.6.4 Exchanging ROAP messages over OBEX

ROAP messages originating from the RI are sent from the Connected Device to the Unconnected Device using the OBEX PUT operation. When receiving a ROAP Trigger that contains the proxy attribute and which is therefore not intended for the Connected Device the Connected Device SHALL maintain the connection to the RI and attempt to establish an OBEX connection to the Unconnected Device's OBEX server (if one does not already exist) and send the ROAP Trigger in an OBEX PUT operation. The Connected Device MUST extract the roapURL from the ROAP Trigger and store for later use. The Connected Device searches for available OBEX servers through service discovery, see Section 14.6.5. In the case where the Connected Device detects multiple OBEX servers (i.e. Unconnected Devices) the Connected Device SHOULD at this time prompt the user to determine which Unconnected Device to connect to.

When receiving a ROAP message in the body of an OBEX response message from the Unconnected Device, the Connected Device SHALL forward the message to the **roapURL** as specified in the ROAP Trigger, possibly re-using the maintained connection to the RI. The Connected Device SHALL close the connection to the RI when the OBEX session ends. The Connected Device MAY close the connection to the RI when receiving a response to a PUT request with response code 0xA0 and no Body (or End of Body) header.

Sending a ROAP message can take one or more OBEX packets. The OBEX server on Unconnected Devices MUST be able to receive multiple sequential PUT requests.

ROAP messages originating from the Unconnected Device are received by the Connected Device in response to PUT operations or by use of the OBEX GET operation (when the response is larger than the maximum OBEX packet length). A Connected Device that has sent a PUT request with the final bit set, and receives a response with response code 0x90 MUST issue GET requests until the complete ROAP message has been received (response code 0xA0).

Each ROAP message MUST be transferred as a ROAP MIME media type within the body of the OBEX request or response. However in order to transfer the message it may split the message into several PUT requests (or GET responses), followed by a PUT Final request (or a final GET response).

14.6.4.1 13.6.4.1 OBEX Response Codes

The OBEX response code contains the HTTP status code (a 3 digit ASCII encoded positive integer) encoded in the low order 7 bits as an unsigned integer as defined in [OBEX]. A Connected Device shall therefore simply translate HTTP status codes in messages received from a Rights Issuer to OBEX response codes before responding to outstanding requests from an Unconnected Device.

14.6.5 Service Discovery

14.6.5.1 IrDA

To enable an OBEX connection over IrDA, the OBEX protocol stack needs to provide IAS setting information to the IAS protocol stack. The Unconnected Device SHOULD use the following IAS entry settings for ROAP communication via OBEX over IrDA:

Class OBEX:ROAP-client

Attribute Name: IrDA:TinyTP:LsapSel

Attribute Type: Integer

Attribute Description: IrLMP LSAP selector for ROAP over IrOBEX, legal values from 0x01 to 0x6F

14.6.5.2 Bluetooth

Service discovery can enhance the user experience by automating selection procedures. This section contains a definition of the corresponding service records and SDP PDUs, needed to enable a Connected Device to automatically find suitable Unconnected Devices to connect to when using the Bluetooth protocol stack.

To enable ROAP over the Bluetooth protocol stack, the Unconnected Device SHOULD advertise service records, which can be retrieved by a Connected Device using the Bluetooth Service Discovery Protocol (SDP).

In the case of the Unconnected Device, the following information, i.e., service records, SHOULD be put into the SDDB (Service Discovery Database):

Item	Definition	Type/ Size	Value	AttrID	Status	Default Value
Service Class ID List			N/A	0x0001**	MUST	
Service Class #0	ROAP unconnected Device	UUID	1d29f667-3236-374a-bf63-3c7a023bb17d	N/A	MUST	
Protocol Descriptor list			N/A	0x0004**	MUST	
Protocol ID #0	L2CAP	UUID	0x0100**	N/A	MUST	
Protocol ID #1	RFCOMM	UUID	0x0003**	N/A	MUST	
Param #0	CHANNEL	Uint8	Varies	N/A	MUST	
Protocol ID #2	OBEX	UUID	0x0008**	N/A	MUST	
Service name	Displayable Text name	String	Varies	0x0000+b***	MAY	“ROAP client”

Table 21: ROAP Client Service Records

** The value or the attribute ID is specified in the Bluetooth Assigned Numbers specification.

*** ’b’ in this table represents a base offset as given by the LanguageBaseAttributeIDList attribute. For the principal language b must be equal to 0x0100 as described in the [Bluetooth SDP] specification.

Table 22 shows the specified SDP PDUs (Protocol Data Units), which are required.

PDU no.	SDP PDU	Ability to Send		Ability to Retrieve	
		ROAP Connected Device	ROAP unconnected Device	ROAP Connected Device	ROAP unconnected Device
1	SdpErrorResponse	N/A	MUST	MUST	N/A
2	SdpServiceSearchAttribute-Request	MUST	N/A	N/A	MUST
3	SdpServiceSearchAttribute-Response	N/A	MUST	MUST	N/A

Table 22: SDP PDUs

14.6.6 Bluetooth Considerations

14.6.6.1 Use of Bluetooth security

Bluetooth authentication and link encryption may be used when running ROAP over OBEX (over Bluetooth). Before these services are available the Connected Device and the Unconnected Device must have gone through an initialisation procedure, i.e. be paired. The initialisation procedure could be a part of the first ROAP session or it could be done in advance if the Connected Device and the Unconnected Device are already paired for other services.

It is expected that Devices in the user's environment are paired once to enable several services.

15.Super Distribution

15.1 Overview

OMA DRM v2 provides two mechanisms for superdistribution:

DRM Content, in the form of a DCF, can be distributed from one Device to another over any physical removable media, wired or wireless network connection without any restrictions.

If the ContentURL header is present within a given DCF (as defined in OMA DCF Specification [DRMDCF-v2.1]), this URL MAY be distributed from one Device to another without any restrictions.

15.2 Preview

If a super-distributed DCF has headers indicating that it supports previews, then the receiving Device MAY use the information provided to generate a preview for the user. This preview may be provided in the form of “instant preview”, where the DCF itself has a preview element that can be used without a Rights Object. In addition, the DCF headers may also indicate a preview method where the Device would need to acquire a preview Rights Object before providing any preview capabilities. See the [DRMDCF-v2.1] for further details on the appropriate DCF headers.

15.3 Transaction Tracking

A DCF may contain a TransactionID as an information element inside an OMADRMTransactionTracking box according to [DRMDCF-v2.1]. A TransactionID refers to an RO acquisition for one or more OMA DRM Containers within a DCF. The TransactionID may be used to track the content flow from one user to another via super distribution from an RI perspective.

The Device MUST ensure the consent of the user for related operations performed by the Device to ensure the privacy issues of the user. This can be done by general settings in the Device, by individual settings per Rights Issuer or on a case by case basis and is implementation specific.

To enable transaction tracking a DCF or PDCF must contain an OMADRMTransactionTracking box when it is received by the Device.

If a DRM Agent receives an RO Response containing an RO and a TransactionID it MUST ask the user for consent to replace the TransactionID in the DCF/PDCF. This can be done in general or on a case-by-case basis. When consent is given, the DRM Agent MUST replace the TransactionID contained in the OMADRMTransactionTracking box of the corresponding DCF or PDCF with the received TransactionID. Otherwise (no consent given) the DRM Agent MUST NOT change the DCF/PDCF. Note: a Device neither needs to generate an OMADRMTransactionTracking box nor needs to change the size of the DCF/PDCF.

If a Device submits an RO Request based on a DCF or PDCF that contains an OMADRMTransactionTracking box it MUST insert the TransactionID of the corresponding DCF or PDCF into the RO Request as the TransactionID,except in the case where the ROAP trigger does not contain a <contentID> because the RO to be delivered is a group or parent RO. The DRM Agent normally SHOULD identify the appropriate DCF by the **<contentID>** element of the ROAP Trigger.

Connected Devices MUST support Transaction Tracking functionality as described above, Unconnected Devices MAY support this feature.

Transaction tracking might not work if

the first user decides to super distribute only the ContentURL instead of the DCF or PDCF or

the Rights Object is received prior to the DCF or PDCF.

Informative Note: The transaction tracking feature may be used by a Rights Issuer to implement a reward mechanism by which a first user may obtain a benefit from super distributing DRM Content to another user which purchases an RO to obtain access to the super distributed content. Transaction tracking comprises the following steps:

a) *The RI provides a TransactionID in an RO Response message to the Device of the first user.*

b) *The Device of the first user replaces the TransactionID in the DCF or PDCF already on the Device with the received TransactionID.*

c) *The first user super distributes the DCF or PDCF to a second user.*

d) *The second user sends an RO Request message including the TransactionID of the received DCF or PDCF to the RI.*

e) *The RI maps the received TransactionID to the same TransactionID related to the initial transaction with the first user.*

The kind of benefit the first and/or second user may get from the RI is out of scope of this specification.

15.4 DCF Integrity

Content in an encrypted DCF/PDCF is immutable. However, a Device MAY add, remove or edit a MutableDRMInformation box in a DCF/PDCF, but in this case the MutableDRMInformation box MUST be located after the last OMADRMContainer box in the DCF or after the movie box in PDCF. Devices MAY add or remove Rights Objects in the MutableDRMInformation box and also edit the OMADRMTransactionTracking box if it is present in the top-level MutableDRMInformation box. The integrity check on the DCF/PDCF MUST be carried out from the beginning until the end of the last OMADRMContainer in the DCF (ignoring the MutableDRMInformation box at the end), i.e. the integrity check is always excluding the MutableDRMInformation box, which MAY be present in the DCF/PDCF. If a DCF or PDCF is modified somewhere outside the ignored structure, the integrity check will fail. This applies to all DCFs whether it is a DCF with one or more DRM Container elements within it or a PDCF with media tracks.

16.Compact Encoding

16.1 Introduction [Informative]

WBXML 1.3 [WBXML] is a simple method that allows compacting XML documents in a loss-less manner. A WBXML decoder processes a WBXML encoded document by interpreting it byte-by-byte. Some bytes represent decoding instructions, some represent XML element start tags, attribute names or attribute values. The decoding process is stateful. The decoder maintains one global state, which determines whether it is processing elements, or attributes. Within each state, the decoder maintains an independent notion of a selected code page.

16.1.1 Code Pages

Code pages in WBXML have 256 entries. There are 256 separate tag code pages for element processing, and 256 separate attribute code pages for attribute processing.

Tag code pages are divided into 4 blocks of 64 entries each. The first 5 entries of each block are reserved for processing instructions. These allow, for example, to switch between tag code pages, signal the literal encoding of an element tag unknown to the WBXML encoder, etc. The remaining 59 entries in each block are associated with a particular XML element tag.

The four blocks each have a specific interpretation. The tag entries in the first block have codes 5 to 63 (inclusive). A code from this range is used to signal an XML element without child elements (i.e. no content), and no attributes. For example: **<tag>**</tag>.

The tag entries in the second block have codes 69 to 127 (inclusive). A code from this range signals an XML element with children elements, but no attributes. For example: **<tag>**<child></child></tag>

The tag entries in the third block have codes 133 to 191 (inclusive). A code from this range signals an XML element without children elements, but with an attribute list. For example: **<tag** attribute=value**>**</tag>

Finally, tag entries in the fourth block have codes 197 to 255 (inclusive). A code from this range signals an XML element with children elements as well as an attribute list. For example: **<tag** attribute=value**>**<child></child></tag>

The attribute code pages are divided in four blocks as well. As with tag code pages, the first 5 entries of each block represent processing instructions. The codes from the first two blocks, with codes 5 to 63 and 69 to 127 all represent attribute names. The codes from the last two blocks, with codes 133 to 191 and 197 to 255 represent attribute values.

16.1.2 String Table

In contexts that use many fixed valued character strings, the WBXML string table is useful. It allows referencing a certain fixed valued string with two bytes. In the OMA DRM context, fixed valued strings are not often used, except as attribute values. Because attribute values have a dedicated representation, the string table is not used.

WBXML Document Format

A WBXML starts with a header signaling:

- the version of the WBXML specification used,
- a document type identifier
- the character set used to encode strings
- a string table.

For this specification, WBXML 1.3 is used. This is signaled by the version byte value 0x03. The document type identifier is different from the one used in OMA DRM v1.0. The identifier “-//OMA//DRM 2.1//EN” is registered by OMNA, and is signaled as the single byte value 0x13.

The character set to be used is UTF-8, signaled as the one byte value 0x6a.

The string table is always empty, this is signaled as four byte values 0x00 (representing a string table length of 0).

Following the header, there is a body. This is a sequence of codes from the code pages, signaling processing instructions, element tags, attribute names and values. The decoder starts with tag code page 0, and attribute code page 0.

16.2 Attribute Code Pages

There is one attribute code page defined. This holds attribute names and attribute values.

“-//OMA/DRM 2.1//EN” – Attribute Code Page 0				
Attribute Name	**WBXML Attribute Code**		**Attribute Value**	**WBXML Attribute Value Code**
xsi:type	5		"urn:oma:bac:dldrm:roap-1.0"	**85**
xmlns:roap	6		"http://odrl.net/1.1/ODRL-EX"	**86**
xmlns:xsi	7		"http://odrl.net/1.1/ODRL-DD"	**87**
xmlns:xenc	8		"http://www.openmobilealliance.com/oma-dd"	**88**
xmlns:ds	9		"http://www.w3.org/2000/09/xmldsig#"	**89**
xmlns:o-ex	A		"http://www.w3.org/2001/04/xmlenc#"	**8A**
xmlns:o-dd	B		"http://www.w3.org/2001/XMLSchema-instance"	**8B**
xmlns:oma-dd	C		"roap:X509SPKIHash"	**8C**
version	D		"http://www.w3.org/2000/09/xmldsig#sha1"	**8D**
proxy	E		"http://www.w3.org/2001/10/xml-exc-c14n#"	**8E**
id	F		"http://www.w3.org/2000/09/xmldsig#hmac-sha1"	**8F**
Id	10		"1.1"	**90**
algorithm	11		"2.0"	**91**
Algorithm	12		**"2.1"**	**92**
URI	13		“identificationRequest”	**93**
type	14		“roUploadRequest”	**94**
			“meteringReport”	**95**
			“leaveDomain”	**96**

All attribute code pages 1 to 127 in the context of the public identifier “-//OMA//DRM 2.1//EN” are reserved for future use by OMA. Within that same context, and within attribute code page 0, attribute name codes from the range 21-3F and 45-7F as well as attribute value codes from the range BA-BF and C5-FF are reserved for future use by OMA.

16.3 Tag Code Pages

16.3.1 ROAP Trigger Context

The element tag code page defined in this section is to be used together with attribute code page 0 to encode and decode ROAP Triggers. Since decoding starts with tag code page 0, this means that for ROAP Triggers, no switch is required to its specific code page.

“-//OMA/DRM 2.1//EN” – Tag Code Page 0

	WBXML Tag Codes			
Tag Name	**No content No attributes**	**Content No attributes**	**No Content Attributes**	**Content Attributes**
roap:roapTrigger	5	45	85	**C5**
registrationRequest	6	46	86	**C6**
roAcquisition	7	47	87	**C7**
joinDomain	8	48	88	**C8**
leaveDomain	9	49	89	**C9**
signature	A	4A	8A	**CA**
encKey	B	4B	8B	**CB**
riID	C	4C	8C	**CC**
riAlias	D	4D	8D	**CD**
nonce	E	4E	8E	**CE**
roapURL	F	4F	8F	**CF**
domainID	10	50	90	**D0**
domainAlias	11	51	91	**D1**
roap:domainID	12	52	92	**D2**
roID	13	53	93	**D3**
roAlias	14	54	94	**D4**
contentID	15	55	95	**D5**
roap:X509SPKIHash	16	56	96	**D6**
keyIdentifier	17	57	97	**D7**
hash	18	58	98	**D8**
ds:SignedInfo	19	59	99	**D9**
ds:SignatureValue	1A	5A	9A	**DA**
ds:KeyInfo	1B	5B	9B	**DB**
ds:CanonicalizationMethod	1C	5C	9C	**DC**
ds:SignatureMethod	1D	5D	9D	**DD**
ds:Reference	1E	5E	9E	**DE**
ds:RetrievalMethod	1F	5F	9F	**DF**
ds:Transforms	20	60	A0	**E0**
ds:DigestMethod	21	61	A1	**E1**
ds:DigestValue	22	62	A2	**E2**
ds:Transform	23	63	A3	**E3**
xenc:EncryptionMethod	24	64	A4	**E4**
xenc:CipherData	25	65	A5	**E5**
xenc:CipherValue	26	66	A6	**E6**
extendedTrigger	**27**	**67**	**A7**	**E7**
trgLeaveDomain	**28**	**68**	**A8**	**E8**
deviceID	**29**	**69**	**A9**	**E9**

In the context of the public identifier “-//OMA//DRM 2.1//EN”, within tag code page 0, tag codes from the range 27-3F, 67-7F, A7-BF and E7-FF are reserved for future use by OMA.

16.4 Processing

16.4.1 MIME Type

ROAP Triggers encoded in WBXML format are identified with the following mime type:

application/vnd.oma.drm.roap-trigger+wbxml (DRM ROAP Trigger)

16.4.2 Binary Data Representation

Binary data is not used in Exclusive Canonical ROAP Triggers. However Rights Issuers MAY use opaque data to represent whitespace in the canonical XML

16.4.3 base64Binary Representation

Some elements in Exclusive Canonical ROAP Triggers hold base64Binary data. Such data is encoded in the WBXML form directly as a literal string. This does not achieve maximum efficiency but intended to simplify the processing of the WBXML decoder.

16.4.4 Rights Issuer

Rights Issuers MAY support WBXML encoding of ROAP Triggers.

Rights Issuers are recommended not to embed whitespace between elements in ROAP Triggers that are to be WBXML encoded.

16.4.5 Device

Devices MUST support WBXML decoding of ROAP Triggers.

If a Device supports WBXML decoding of ROAP PDUs, it must signal this in the HTTP Accept header of any of its requests by including the applicable mime type (application/vnd.oma.drm.roap-trigger+wbxml).

16.4.6 Normal Processing and Transcoding

After receiving a message in WBXML format, and if that message in WBXML format is supported by the recipient, that recipient MUST decode the message into Exclusive Canonical XML format before any other processing is applied.

17. Export

17.1 Introduction

After downloading OMA DRM Content, the User may wish to consume that content on another Device that has a different DRM protection format. Export is an operation in which the DRM Content and corresponding rights are transferred to a DRM system or content protection scheme other than the OMA DRM system. The Rights Issuer controls whether or not to allow the export.

The Rights Issuer must explicitly grant permission (with the <export> element in [DRMREL-v2.1]) before the content and Rights Object can be exported. The Rights Issuer also specifies to which DRM system or content protection scheme the DRM content is allowed to be exported. The Rights Issuer MAY permit export to more than one system.

Only the basic concept of export from OMA DRM to another DRM system or content protection scheme is specified in this document. OMA does not specify the exact rules for transcribing Rights Objects to the other protection mechanisms. It is the responsibility of appropriate bodies governing the use of those protection systems to define the necessary mechanisms for transcribing OMA DRM Rights Objects. Figure below explains the principle.

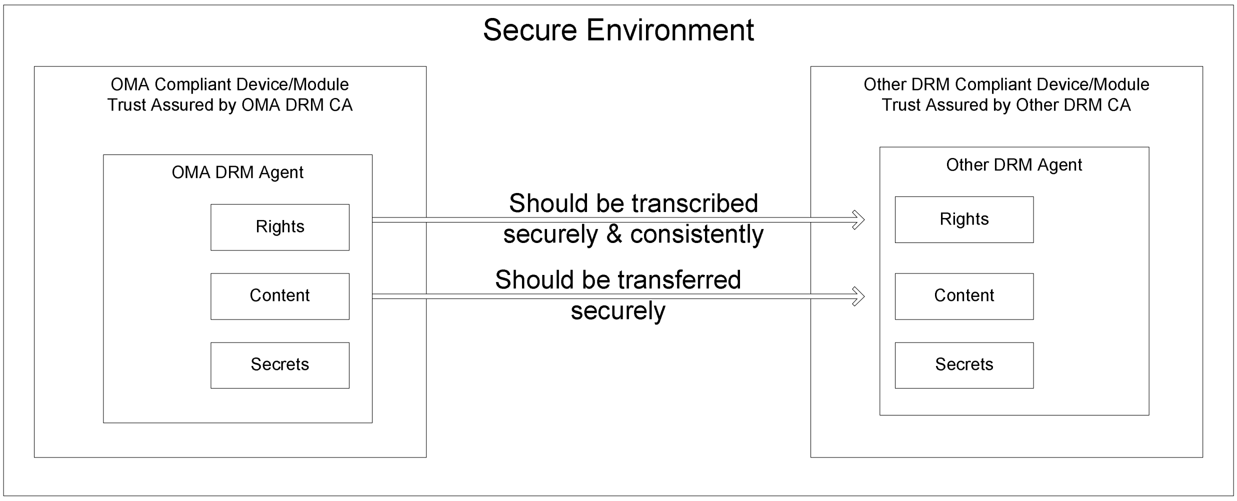

Figure 19: Exporting from OMA DRM

17.2 Export Modes

The Rights Issuer can specify if the DRM content and Rights Object are available on the original Device after the export (“copy”) or are permanently removed following the export (“move”).

In the case of “copy”, the DRM Content and Rights Object remain on the original Device and are available for consumption following the export. The Rights Issuer MAY specify the number of times the “copy” export is permitted. The original Rights Object is exported without state information if it is a stateful Rights Object, it MUST remain unchanged on the original Device after the export.

In the case of “move”, the original Rights Object MUST become permanently unusable on the original Device, after exporting is conducted. The Rights Object MUST be exported with the current state information at the time of the export if it is a stateful Rights Object. That is, if a stateful right has been partially consumed, only the remaining portion is exported. The Content Object MAY remain on the original Device.

In either mode, the <export> permission MAY be transcribed into target DRM system or content protection scheme as indicated by the “transcribe” attribute of the <export> element.

17.3 Compatibility with Other DRM Systems

The targeted DRM system may not support all of the capabilities of OMA DRM. Some potential areas of incompatibility include:

- Content Types
- OMA REL usage permissions and constraints
- Multiple Rights Objects for a single content
- Rights for multiple content objects in a single Rights Object

This section defines some general rules to minimize incompatibilities when exporting to non-OMA DRM systems. The detailed rules for the transcription of OMA Rights Objects to those of another DRM system are specific to the target system and, therefore, are not part this document.

During discovery and download of content for future export, the best possible content and rights should be provided to the Device according to device capability, the capability of the other DRM system, and user preferences. This information MAY be indicated to the Content Issuer using [UAProf] as specified in section 13.3.

When creating a Rights Object for Export (i.e. **<**export**>** permission is included), the Rights Issuer SHOULD construct the Rights Object so that all the permissions and constraints within it are supported by the other DRM system. All permissions and constraints in the original Rights Object MUST be transcribed provided they are supported in the target DRM system. The <export> with mode “move” MUST NOT be present in a parent RO or a group RO.

As described in section 10.2, a single Rights Object can contain rights for multiple content objects either within a multipart DCF or separate (P)DCFs. The <export> permission is applied to the entire Rights Object so that when such a Rights Object is exported, each associated content object MUST also be exported except when <export> with mode “copy” occurs in a parent RO or a group RO. For a parent RO, the export copy enables export of any content object referenced by a related child RO and for a group RO, export copy enables export of any content object within the group. In the case of multi-asset RO then all content objects references by that multi-asset RO MUST be exported. When exporting multi-track PDCFs or multi-part DCFs then, all tracks and content objects within the multipart MUST be exported if permitted by the associated RO.

A single content object may have more than one corresponding Rights Object. If the user wishes to export this content object, all Rights Objects with permission to export to the targeted DRM system MUST also be exported. If the target DRM system supports multiple rights for a single content object, multiple rights in the original Rights Object MUST be transcribed. If the target DRM system does not support multiple rights for a single content object, the multiple rights MAY be merged into one Rights Object and then transcribed.

17.4 Streaming to Other Devices

Another form of export allows the user to stream DRM content from the original Device to a rendering Device (i.e. headphones) for immediate playback. The content MUST be streamed over a copy protected medium where the transmission protocol between the Devices ensures that the DRM content cannot be copied in an unauthorised manner.

The general rules above in terms of transcribing the content and rights SHOULD be followed when streaming over protected links for rendering purposes.

When <export> permissions are granted and the target system is a link protection scheme, it is understood that a transient copy is made to facilitate rendering on the target Device. The appropriate signalling MUST be used to indicate to the target DRM/protection system, that the streamed content is used only for rendering purposes.

18.Unconnected Device Support

The following section identifies how a Connected Device can act as an intermediary to assist an Unconnected Device to purchase and download content and Rights Objects. This Functionality enables, for example, a portable mobile Device that does not have inherent network connectivity to acquire content and associated rights. This functionality builds on the Domain concept as described in section 7.

An Unconnected Device SHALL be capable of connecting to a Rights Issuer via a Connected Device using an appropriate protocol over a local connectivity technology. E.g. OBEX over IrDA, Bluetooth or USB as specified in section 8.

Unconnected Devices MUST support the 4-pass Registration protocol as specified in section 5.4.2.

Unconnected Device MUST support Domain Join and Domain Leave protocols as specified in section 5.4.5.

Unconnected Devices that do not support DRM Time SHALL NOT send RO Acquisition Request messages as specified in section 5.4.4.1.

A Connected Device MAY also implement Unconnected Device functionality.

Connected Devices SHALL be capable of directly connecting to a Rights Issuer using an appropriate protocol over an appropriate wide area transport/network layer interface (e.g. HTTP over TCP-IP).

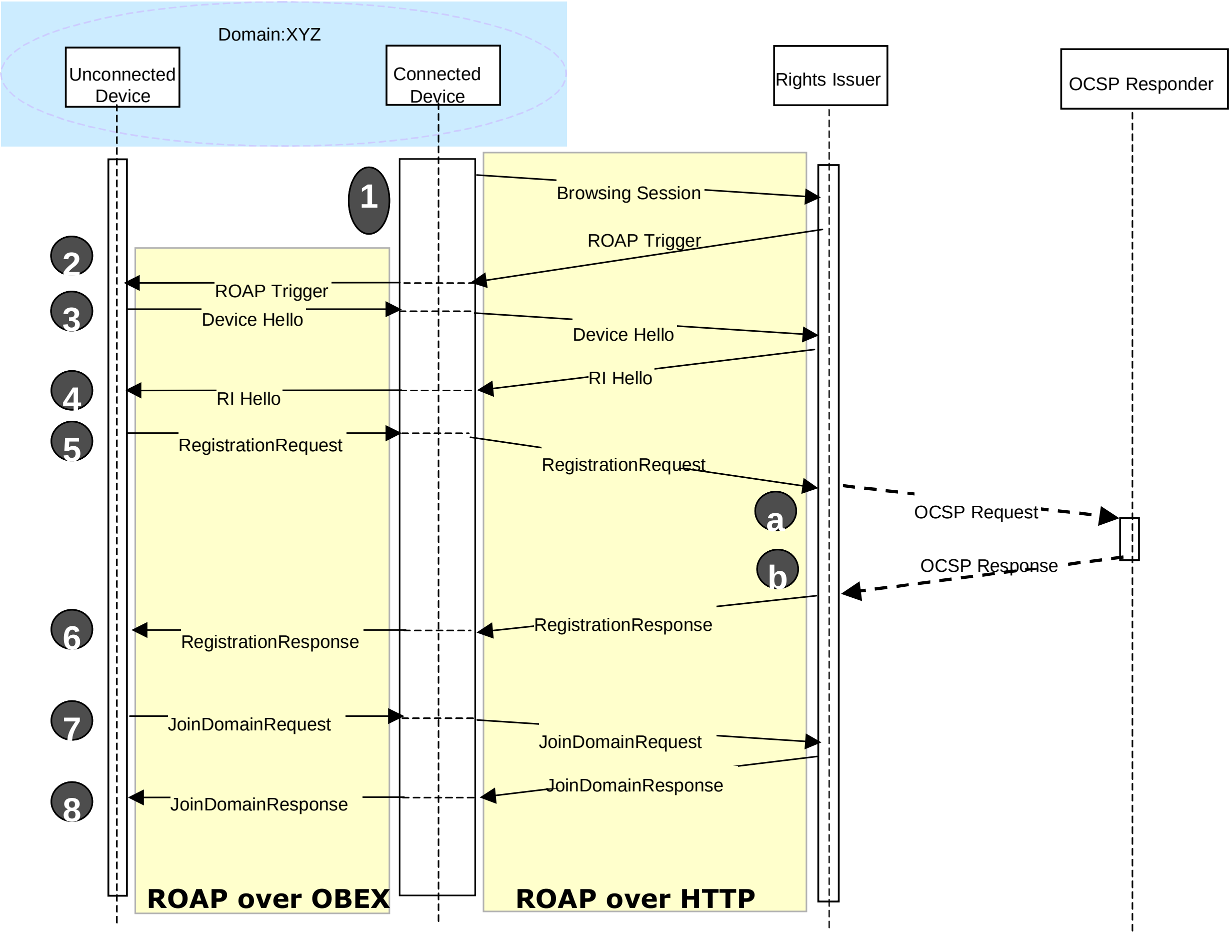

Figure 20: Unconnected Device Registration and Domain Establishment

The above diagram shows how an Unconnected Device establishes an RI Context (Registration) and is added to the Domain: XYZ. In the above diagram it is assumed that the Connected Device has already performed the required steps in order to join the Domain: XYZ.

The user initiates a browsing session from the Connected Device to an RI. The user indicates to the RI that they would like to add an Unconnected Device to the Domain: XYZ (how this is achieved is outside of the scope of this specification).

The RI returns a ROAP Trigger of type joinDomain to the Connected Device and includes the proxy attribute with the value set to “True”.

Upon receipt of the ROAP Trigger, the Connected Device determines that the ROAP Trigger is intended for an Unconnected Device (through examination of the proxy attribute). At this point the Connected Device SHALL maintain the connection to the RI and attempt to establish an OBEX connection to the Unconnected Device's OBEX server. Once the OBEX connection is established the Connected Device MUST send the ROAP Trigger in an OBEX PUT operation. The Connected Device MUST extract the roapURL from the ROAP Trigger and store for later use.

If it is not clear to the Connected Device which Unconnected Device should receive the ROAP trigger (e.g. because the Connected Device is already in communication with more than one Unconnected Device) then the Connected Device SHOULD display a list of the Unconnected Devices to the user (using user friendly identifiers for the Unconnected Devices) and request the user to select the Unconnected Device that the user wishes to be joined to the Domain). The Connected Device MUST attempt to establish an OBEX connection to the OBEX server of the user-selected Unconnected Device.

Upon reception of the joinDomain ROAP Trigger the Unconnected Device MUST determine whether it has an RI context with the RI or not. If the Unconnected Device does not have an RI context with the RI indicated in the ROAP trigger the Unconnected Device MUST send a ROAP-DeviceHello message in the OBEX response to the Connected Device. The Connected Device SHALL forward the message to the roapURL as specified in the ROAP Trigger, re-using the maintained connection if possible. If the Unconnected Device has an RI Context then steps 5-7 do not apply.

The RI MUST respond with a ROAP-RIHello message which the Connected Device MUST send to the Unconnected Device’s OBEX server in an OBEX PUT operation

The Unconnected Device MUST respond with a ROAP-RegistrationRequest message in the OBEX response to the Connected Device. The Connected Device SHALL forward the message to the roapURL as specified in the ROAP Trigger re-using the maintained connection.

The RI MUST respond with a ROAP-RegistrationResponse message which the Connected Device will send to the Unconnected Device’s OBEX server in an OBEX PUT operation

Upon successfully establishing an RI context, the Unconnected Device MUST send a ROAP-JoinDomainRequest message in the OBEX response to the Connected Device. The Connected Device SHALL forward the message to the roapURL as specified in the ROAP Trigger, re-using the maintained connection.

The RI MUST respond with a ROAP-JoinDomainResponse message which the Connected Device MUST send to the Unconnected Device’s OBEX server in an OBEX PUT operation.

The Unconnected Device MAY respond with an OBEX Disconnect or MAY respond with an OBEX message containing the code 0xA0 and no Body (or End of Body) header. Upon reception of the OBEX Disconnect operation, the Connected Device SHALL close the connection to the RI. The Connected Device MAY close the connection to the RI when receiving a response to a PUT request with response code 0xA0 and no Body (or End of Body) header.

In the above diagram and text it is assumed that no ROAP specific errors occur during the ROAP session. If ROAP specific errors occur during the ROAP session then the Unconnected Device SHOULD use the value of the status parameter as defined in section 5.3.6 and act accordingly.

Once an Unconnected Device has successfully registered and joined the same Domain as the Connected Device then the Connected Device can acquire content and rights on behalf of the Unconnected Device. RO acquisition is shown below.

In the case of Unconnected Device, for silent Device Registration or silent Domain Join (include Upgrade), the Unconnected Device MAY need to indicate to user that he/she needs to use a Connected Device to initiate a Device Registration or Domain Join (i.e. silent Device Registration or silent Domain Join is not allowed for Unconnected Device).

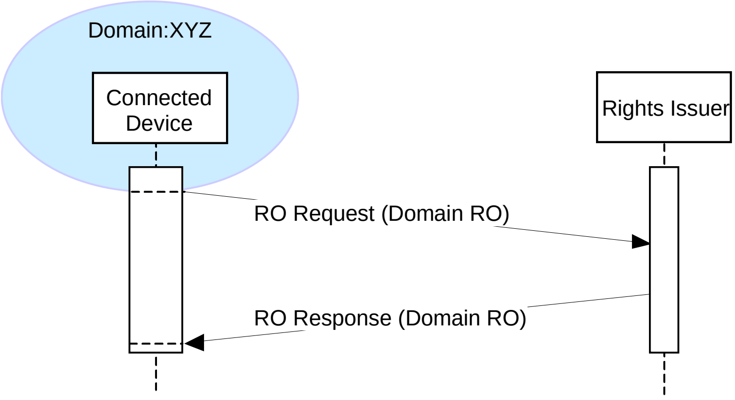

Figure 21: RO Acquisition

Once the Connected Device has received the Domain RO, it SHOULD insert the Domain RO in the associated DCF as specified in section 8.7.2.2. This enables the DCF and embedded RO to be transferred to the Unconnected Device using a simple file transfer operation over OBEX as shown in the following diagram.

Note: As an Unconnected Device may not support the acquisition of rights, a Connected Device should present a suitable warning to the user, if the user attempts to transfer a DCF without an embedded Domain ROs to an Unconnected Device.

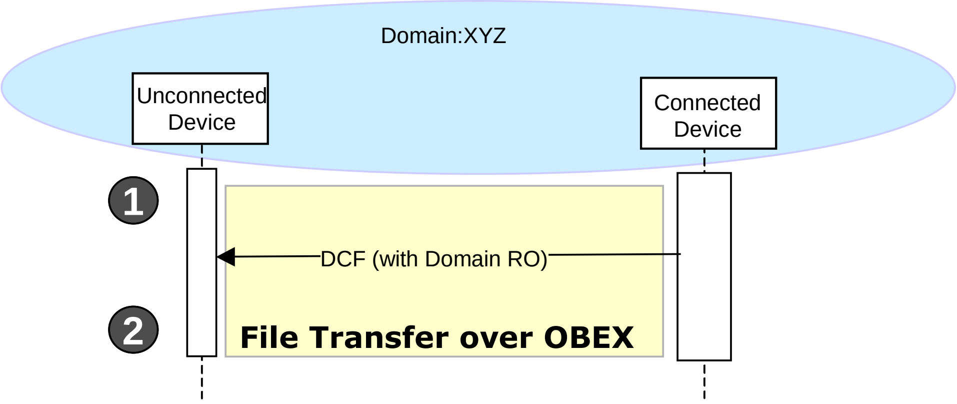

Figure 22: Content Acquisition

In the case where a user wishes to remove an Unconnected Device from a Domain, this is achieved as follows:

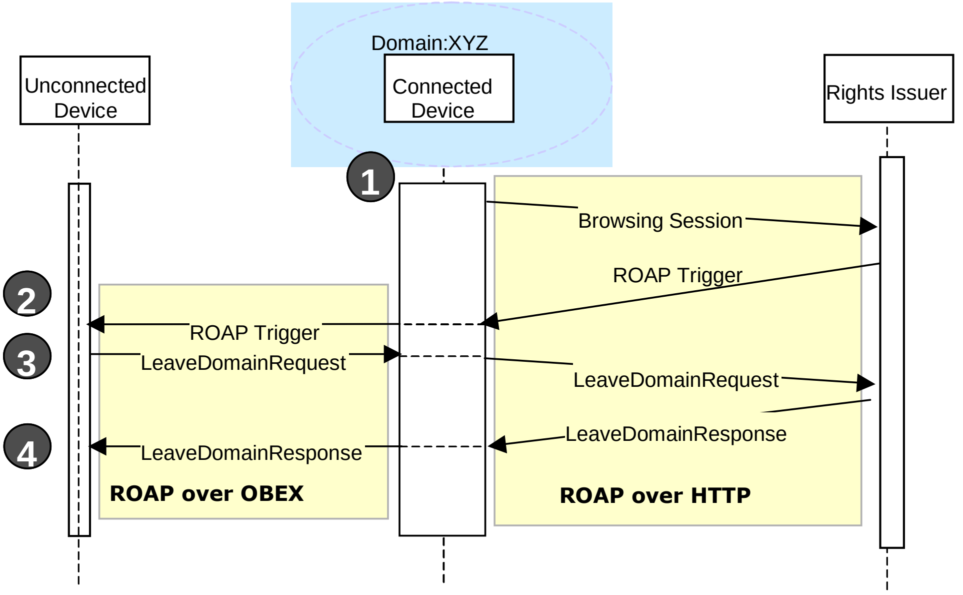

Figure 23: Unconnected Device leaving a Domain

The user initiates a browsing session from the Connected Device to an RI. The user indicates to the RI that they would like to remove an Unconnected Device from the Domain: XYZ (how this is achieved is outside of the scope of this specification). The RI returns a ROAP Trigger of type leaveDomain to the Connected Device and includes the proxy attribute with the value set to “True”.

Upon receipt of the ROAP Trigger the Connected Device determines that the ROAP Trigger is intended for an Unconnected Device (through examination of the proxy attribute). At this point the Connected Device SHALL maintain the connection to the RI and attempt to establish an OBEX connection to the Unconnected Device's OBEX server and sends the ROAP Trigger in an OBEX PUT operation. The Connected Device MUST extract the roapURL from the ROAP Trigger and store for later use.

Upon reception of the leaveDomain ROAP Trigger and after performing the steps specified in section 8.4 the Unconnected Device MUST send a ROAP-LeaveDomainRequest message in the OBEX response to the Connected Device. The Connected Device SHALL forward the message to the roapURL as specified in the ROAP Trigger re-using the maintained connection.

The RI will respond with a ROAP-LeaveDomainResponse message, which the Connected Device will send to the Unconnected Device’s OBEX server in an OBEX PUT operation.

The Unconnected Device MAY respond with an OBEX Disconnect or MAY respond with an OBEX message containing the code 0xA0 and no Body (or End of Body) header. Upon reception of the OBEX Disconnect operation the Connected Device SHALL close the connection to the RI. The Connected Device MAY close the connection to the RI when receiving a response to a PUT request with response code 0xA0 and no Body (or End of Body) header.

19.Binding Rights to User Identities

19.1 IMSI uid

If the Device supports a SIM/USIM/R-UIM and the <uid> element of a child <context> element of an <individual> element within a Rights Object specifies an IMSI, the DRM Agent MUST observe the following behaviour.

When the associated content is selected for rendering the DRM Agent MUST check that the IMSI on the currently installed SIM/USIM/R-UIM (as stored within EF$_{\_IMSI}$ elementary file, which is defined in [3GPP TS 51.011], [3GPP TS 31.102] and [3GPP2 C.S0023-B]) matches the IMSI specified within the <uid> element or in the case where the RO is bound to multiple IMSIs one of the IMSI's specified within the <uid> element.

If the IMSI of the currently installed SIM/USIM/R-UIM matches the value (or in the case when the RO is bound to multiple IMSIs one of the values) specified in the <uid> element, as specified in [DRMREL-v2.1], then the <permission> SHOULD be exercised if all other constraints are fulfilled.

If the IMSI of the currently installed SIM/USIM/R-UIM does not match the value (or in the case where the RO is bound to multiple IMSI's any of the values) specified in the <uid> element then the <permission> MUST NOT be exercised.

19.2 WIM uid

If the Device or SIM, UICC in the Device supports a WIM and if the Device supports binding to WIM and, if the <uid> element of a child <context> element of an <individual> element within a Rights Object specifies a PKC_ID, the DRM Agent MUST observe the following behaviour.

For signature verification, the Device attempts to retrieve the user certificate from the WIM [WIM], identified by PKC_ID, i.e. checking if the value of PKC_ID matches with the value of CommonCertificateAttributes.certHash field from the user certificate CDF entry,

Compute a hash (e.g. thumbprint) over the user certificate. The hash is calculated over the DER encoding of the complete certificate and SHA1 hashing algorithm MUST be used,

Check that the hash (e.g. thumbprint) matches the value of the PKC_ID,

Go to step 4 if the result of the check is successful. If unsuccessful, the permission MUST NOT be exercised.

Generate a 20 bytes challenge value,

Ensure that the rights to access the WIM signature key are granted,

Request the WIM to sign the challenge using the private key associated with the identified user certificate,

Verify the signature using the user certificate.

If the verification of the signature is successful then the <permission> SHOULD be exercised if all other constraints are fulfilled. If the verification of the signature failed then the <permission> MUST NOT be exercised.

19.2.1 Support for WIM uid

If the Device or SIM, UICC in the Device supports a WIM and if the Device supports binding to WIM then the DRM Agent SHOULD support User certificate for authentication as defined in Appendix E.

The said user certificate has to be stored locally in the WIM. The logical record of the WIM CDF that provides information for that certificate thus makes use of the path identifier reference choice. In addition, it has to contain the optional field

CommonCertificateAttributes.certHash. The use of the private key associated to the said user certificate has to be protected by the PIN-G i.e., the logical record of the WIM PrKDF that corresponds to that key provides a commonObjectAttributes.authId field that identifies the PIN-G authentication object.

To optimise (i.e. save certificate hashing operation) the next procedures that make use of the said user certificate, it is RECOMMENDED that once the DRM Agent successfully passed the step 3 it stores the trusted couple (user certificate, user certificate hash (e.g. thumbprint)) in its local storage area. Thus, the DRM Agent MAY resume the sequence, starting from step 4 and making use of the user certificate from its local storage area to perform step 7 i.e., selected by PKC_ID == user certificate hash (e.g. thumbprint).

Interactions between the DRM Agent and the WIM are described in Appendix F.

20.Security Considerations (Informative)

20.1 Background

DRM solutions in general need to meet a number of security requirements. In particular, two requirements any DRM solution must fulfill are:

DRM Content must only be accessed by properly authenticated and authorised DRM Agents

Permissions on DRM Content must be honoured by all DRM Agents

This specification along with its accompanying documents ([DRMARCH-v2.1], [DRMREL-v2.1], and [DRMDCF-v2.1]) establishes the OMA DRM system. The OMA DRM system provides the means for the secure distribution and management of DRM Content in the OMA environment, and meets the requirements specified above and in [DRMREQ-v2.1].

20.2 Trust Model

The OMA DRM trust model is built on a PKI. A Rights Issuer trusts a DRM Agent to behave correctly if the DRM Agent's certificate is verifiable by the Rights Issuer and not revoked. Similarly, a DRM Agent trusts a Rights Issuer to behave correctly if the Rights Issuer's certificate is verifiable by the DRM Agent and not revoked.

Devices and Rights Issuers may support multiple PKIs. This means that Devices and Rights Issuers may have multiple certificate chains and/ or multiple trusted root certificates installed.

20.2.1 RIs supporting multiple PKIs

If Rights Issuers are using multiple certificate chains signed by different PKIs the same private/ public key pair MUST be used for each of those PKIs. This leads to one unique RI ID for the Rights Issuer.

Rights Issuers may have additional trusted root certificates from other PKIs installed without having a key pair signed by one of those PKIs. Such a trusted root certificate can be used to validate Device certificate chains issued under that root of trust. For this scenario it is required that the Device trusts one of the roots the server has certificate chains for.

To support multiple RI IDs and multiple public keys it is suggested to setup multiple RIs using different RI URLs with different Fully Qualified Domain Names. This can also be useful to control under which trust model content may be distributed.

20.2.2 Devices supporting multiple PKIs

Devices may have multiple and different key pairs signed by multiple PKIs, what means that they may have multiple Device IDs.

Devices may have additional trusted root certificates from other PKIs installed without having a key pair signed by one of those PKIs. Such a trusted root certificate can be used to validate RI certificate chains issued under that root of trust. For this scenario it is required that the RI trusts one of the roots the Device has certificate chains for.

20.3 Security Mechanisms in the OMA DRM

20.3.1 Confidentiality

Confidentiality ensures that data is not accessible by an unauthorised party. As stated above, DRM Content must only be accessible by properly authenticated and authorised DRM Agents. To achieve this goal, DRM Content is encrypted.

Encryption keys are unique to each Media Object, and Rights Objects carry the encryption keys wrapped in keys only accessible by the intended recipients.

20.3.2 Authentication

Authentication is the process by which a party identifies itself to another party. In the OMA DRM, mutual DRM Agent and Rights Issuer authentication is achieved in the 4-pass Registration Protocol, the 2-pass RO Acquisition protocol, the 2-pass Join Domain protoco, the 2-pass Metering Report protocol, and the 2-pass RO Upload protocol l. Depending on protocol and message, the authentication is achieved either through digital signatures on nonces or time stamps. The 1-pass RO Acquisition protocol achieves Rights Issuer authentication through the digital signature on a time stamp. It does not authenticate the DRM Agent to the Rights Issuer, but due to the DRM Content being wrapped with the recipient's public key, the initial requirement of Section 20.1 is still met. In the 3 or 4-pass confirmed RO Acquisition protocols, authentication is achieved in the confirmation part of the protocol (the last two messages) via a signature on a nonce in each message. The 2-pass Leave Domain Protocol authenticates the DRM Agent to the Rights Issuer through the digital signature on a time stamp. It does not authenticate the Rights Issuer to the DRM Agent.

20.3.3 Integrity Protection

Data integrity protection ensures the ability to detect unauthorised modification of data. In the OMA DRM, data integrity protection, when applicable, is achieved through digital signatures on ROAP messages and Rights Objects.

20.3.4 Key Confirmation

Key confirmation ensures the recipient of a message containing a protected key that the sender of the message knows the key value. In the context of DRM, this property protects against unauthorised re-issuance of Rights Objects from one Rights Issuer by another. In the OMA DRM system, key confirmation is achieved through a MAC over the protected key and the sending party's identity, using parts of the protected key as the MAC key.

20.3.5 Other Characteristics

20.3.5.1 DRM Time

The OMA DRM system assumes the presence of DRM Time in the DRM Agent. Since users are not able to change DRM Time, this specification defines a mechanism by which the DRM Time can be synchronised with the time held by an OCSP responder.

Devices supporting multiple trust models MUST maintain one DRM time per trust model. This is to minimize the effect of a “fake” or compromised trust model on other trust models.

20.3.5.2 Transport Layer Security

The OMA DRM system provides application-layer security through the use of the security mechanisms listed in Section 20.3. Hence, it does not rely on, or assume, transport-layer security

20.3.5.3 Pseudorandom Number Generators

The use of nonces in the OMA DRM system requires DRM Agents and RIs to have pseudorandom number generators of good quality.

20.4 Threat Analysis

20.4.1 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 unauthorised behaviour. 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.

20.4.2 Active Attacks

20.4.2.1 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.

For the Registration protocol, message removal will result in a failed protocol run and no establishment of an RI Context in the Device.

For the RO Acquisition protocol, message removal will result in the non-delivery of the requested Rights Object(s) to the DRM Agent. To ensure correct billing in such a situation, the mechanisms outlined in Section 14.3 may be used (although it is important to note that the suppression of the DLOTA InstallNotify message may reverse the threat – i.e. cause the RI to believe that the Rights Object did not get installed even though it was). In the confirmed RO acquisition protocol, message removal can also result in the non-delivery of installation confirmation information to the RI. Each RI must determine suitable billing behaviour in this situation, e.g. some RIs may resend the ROAP-ROResponse if the ROs contained therein are stateless while others will assume that the confirmation message was removed and bill regardless.

Whenever a ROAP trigger is sent from an RI to a Device, the RI can insert a nonce in the trigger which the Device must insert in the **roap:Request** as a response from the Device to the RI. This allows coupling of triggers and their corresponding **roap:Request** by the RI but also allows the RI, if they choose, to resend the trigger if they do not receive the corresponding **roap:Request** within a certain time period. This allows defeat of non-persistent message removal attacks.

For the Join Domain protocol, if an attacker removes the ROAP-JoinDomainResponse message, a Device will not become a member of the requested Domain even though the RI may think it has. Again, mechanisms outlined in Section 14.3 may ensure delivery but suppression of DLOTA InstallNotify messages may reverse the threat.

For the Leave Domain protocol, if an attacker removes the ROAP-LeaveDomainRequest message from the communication channel before it has reached the RI, the RI may still view the DRM Agent as a member of the Domain. It is important to note the consequences this may have for any billing scheme based on Domain membership.

For the Metering Report protocol, if an attacker removes the ROAP-MeteringReportSubmit message, the RI will not receive the Metering Report.

For the RO Upload protocol, message removal will result in a failed protocol run and no RO uploaded to the RI.

Removal of a ROAP trigger before it reaches the Device will stop the intended ROAP protocol from being executed.

20.4.2.2 Message Modification

An attacker may modify any message sent between a DRM agent and an RI.

For the Registration protocol, the RO Acquisition protocols, the Join Domain protocol, the Metering Report protocol, and the RO Upload protocol, message modification will be detected through the use of digital signatures.

For the Leave Domain protocol, modification of the ROAP-LeaveDomainRequest message will be detected by the RI through the Device's digital signature on the message. The DRM Agent, however, may not detect modification of the ROAP-LeaveDomainResponse message since the message is not digitally signed. This may result in a DRM Agent assuming the RI has removed it from the requested Domain when in fact it has not or vice versa (the DRM agent assuming the RI has not removed it from the requested Domain when in fact it has). The former attack's possible implications for billing schemes should be noted. The latter will result in re-tries by the DRM Agent or the DRM Agent notifying the user.

For the various ROAP triggers, message modification will be detected if the message was signed. In particular, the RI must sign the **<leaveDomain>** trigger. For other triggers, the Device may not detect message modifications.

20.4.2.3 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.

The Registration protocol protects against replay attacks through the use of nonces, ensuring to both parties that the other party is "live".

The 2-pass RO Acquisition protocol assures the DRM Agent that the RI is live through the use of the Device nonce. It assures the RI that the DRM Agent is live through the DRM Agent's signature on the DRM time.

The 1-pass RO Acquisition protocol assures the DRM Agent that the RI is live through the signature on the RI's current time.

The 3 or 4-pass confirmed RO Acquisition protocol assures the DRM Agent that the RI is live through the use of the Device nonce. It assures the RI that the DRM Agent is live through the DRM Agent's signature on the DRM time.

The Join Domain protocol protects against replay attacks in the same way as the 2-pass RO Acquisition protocol.

The Leave Domain protocol assures the RI that the DRM Agent is live through the DRM Agent's signature on the DRM time. It does protect against replay attacks through the use of the Device Nonce (it does not protect against message insertion, however and as noted above).

The Metering Report protocol and the RO Upload protocol protect against replay attacks in the same way as the 2-pass RO Acquisition protocol.

ROAP triggers may be sent to any Device at any time. With the exception of the **<leaveDomain**> and *Extended Leave Domain* (see section 5.2.1.1) triggers, ROAP triggers are not protected against replay attacks. Devices can protect against replay of **<leaveDomain>** or *Extended Leave Domain* triggers due to the use of a mandatory **<nonce>** element. However even with the **<nonce>** a leave domain trigger can be distributed to "another" Device and that will force the target Device to remove its Domain Context. Prevention against this attack is provided in the *Extended Leave Domain* trigger which includes a **<deviceID>** element; thus ensuring the trigger is bound to a specific Device.

All ROAP triggers sent from an RI to a Device can include a nonce within the trigger which the Device must insert in the **roap:Request** sent in response to the trigger. This stops a man in the middle from recording a **roap:Request** sent in response to one ROAP trigger and replaying it as the response to another ROAP trigger.

Protection against replay of stateful ROs and against replay of stateless ROs that have been uploaded to an RI is achieved by means of the method specified in Section 10.4. It is important to note that the replay cache MUST be integrity-protected by the DRM Agent.

20.4.2.4 Denial-of-Service Attacks

Denial-of-Service (DoS) attacks are attacks against the availability of a system and include attacks that consume resources (bandwidth, storage) and/or the destruction of resources (e.g. software or hardware components, data destruction and physical destruction). Such attacks can result in significant loss of revenue and intellectual property.

As an example, an attacker could send multiple ROAP-RORequest messages to the RI, whether authentic or not. These messages may bind significant resources at the RI site (e.g. signature verifications), rendering the RI unable to respond to other requests.

RIs need to implement standard DoS precautions to protect against these attacks, see e.g. [DDOS].

20.4.2.5 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:

- The DRM Agent's private key
- Domain keys for any Domain the DRM Agent is a member of
- Rights Object Encryption Keys
- Content Encryption Keys
- DRM 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.

A compromised DRM Agent may be able to modify a Device RO in such a way that the Rights Object can be re-issued and accepted by an uncompromised DRM Agent. This threat occurs only if the Rights Object contains the <**signature**> element in the **roap:ROPayload**. A suggested method to limit this threat is to include an <**individual**> constraint in Device Rights Objects containing a <**signature**> element in the **roap:ROPayload**.

In addition, a compromised rendering application in the DRM Agent may also result in the loss of DRM Content. The DRM Agent implementation must therefore be robust and ensure that it only provides unprotected DRM Content to trusted rendering applications.

A compromised Rights Issuer may result in the disclosure of any of the following:

- The Rights Issuer's private key
- Domain keys for any Domain administered by the RI
- Rights Object Encryption Keys
- Content Encryption Keys
- DRM 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 [OCSP].

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.

20.4.3 Passive Attacks

An attacker may eavesdrop on and record any conversation carried out between an RI and a DRM Agent. Such eavesdropping may allow the attacker to trace user behaviour and, to some extent, interests. Due to the security features of the OMA DRM system, it will however not allow the passive attacker to perform later off-line attacks against DRM Content, wrapped keys, or Rights Objects exchanged in such recorded messages.

20.5 Privacy

Privacy is the right of an individual to control or influence the amount of information about her collected by others. The data that should be protected can be divided into personal data and interest data. As an example, a content issuer collecting content download data can adapt its offers to the downloaded party according to perceived demands. On one hand this may achieve user convenience, but on the other hand it limits the right of self-determination of the individual. Furthermore, the keeping of log files that store information, such as who has done what at which time, reveals user behaviour and might not be in the interests of the system's users.

The ROAP protocol has no explicit measures to anonymize the association between DRM Agent, downloaded DRM Content and associated RO, since the RI and CI are either administered by one organisation or might exchange information freely. Also, the messages that are sent do not protect the identities of the communicating parties.

In addition, the Transaction Id mechanism to track super distribution behaviour, e.g. for reward purposes, might be perceived as privacy intruding. For instance, a CI inserting values in the Transaction Tracking box of a DCF with the purpose of rewarding customers for super distributing content, potentially learns a great deal of information about users’ forwarding behaviour. In this case, however, the OMA DRM system does allow user policies to determine whether to use the Transaction ID mechanism or not. The Metering report mechanism which is used to collect Metering Information might also affect users’ privacy, since such Metering Information reflects users’ consuming behaviour. OMA DRM providers should consider user's privacy demands to the extent possible and whether anonymity mechanisms can be deployed within the bounds of the OMA DRM system.

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
OMA-TS-DRM-DRM-V2_1-20081014-A	14 Oct 2008	Status changed to Candidate by TP ref# OMA-TP-2008-0382-INP_DRM_V2_1_ERP_for_Final_Approval

Appendix B. ROAP Schema

The ROAP and ROAP trigger XML schema is defined in normative [DRMROAPXSD-v2.1]

Appendix C. Backward Compatibility with Release 1.0

Devices that support OMA DRM v2 MUST support the mandatory features of OMA DRM V1 [DRM]. To ensure consistent, interoperable behaviour, OMA DRM v2 Devices MUST behave in the following manner when receiving OMA DRM v1 Content.

DRM v2 Client receives the following DRM v1 content type	DRM v1 method not supported	DRM v1 method is supported
Forward Lock content	n/a (DRM v1 Forward Lock is mandatory)	Handle content as defined in [DRM]
Combined Delivery content	Handle content as defined in [DRM]	Handle content as defined in [DRM]
Separate Delivery Content	MAY notify the user	Handle content as defined in [DRM]; Upon contacting the CI/RI the Device MUST advertise DRM version and supported media types as defined in section 11.

Table 23: Backward Compatibility with Release 1.0

Appendix D. Application to Services (Informative)

D.1 Application to Streaming Services

The main scope of OMA DRM is protection of downloadable objects, which can by their nature be embedded into DCFs and be delivered under DRM control. This is not immediately possible with streaming media, since streaming media are transported using protocols and mechanisms that do not allow embedding into download DCFs, and also since streams are not per se limited in time and size. Thus, the protected transport of streams and some associated signaling has to be defined separately for streaming media. On the other hand, OMA DRM ROs can be used for streaming services for the definition and transport of rights/permissions, and of content decryption keys.

Thus, the basic concept for the application of OMA DRM to streaming services is that OMA DRM ROs, and the ROAP, are used in the same way as for downloadable objects/DCFs. This is specified in this standard. The exact way of protecting streams, storing streams at a streaming server, and transporting streams to a Device (including associated signaling) are not specified in this specification. It is the responsibility of streaming standardisation bodies to define appropriate mechanisms that work seamlessly together with the concept laid out in the DRM specification, especially with the RO concept and format. Figure 24 explains the principle.

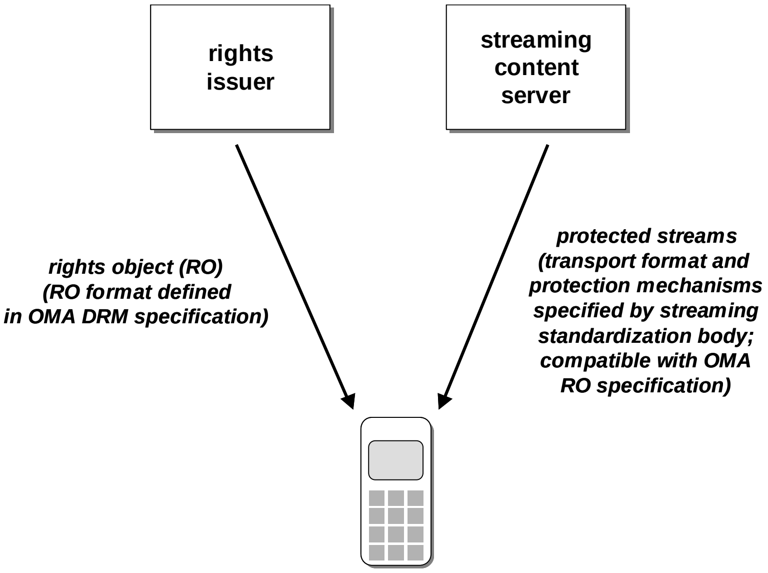
rights issuer
streaming content server
rights object (RO) (RO format defined in OMA DRM specification)
protected streams (transport format and protection mechanisms specified by streaming standardization body; compatible with OMA RO specification)

Figure 24: Generic principle of application of OMA DRM to streaming services

D.1.1 Application to the 3GPP Packet-Switched Streaming Service

For the special case of the 3GPP Packet-Switched Streaming Service (PSS) Release 6, i.e., the 3GPP streaming standard [3GPP PSS], OMA and 3GPP have been working together to define DRM protection of PSS media. The basic principle is the one shown in Figure 24, but there are some extensions that consider special features and properties of the PSS standard, namely

PSS sessions can consist of a mixture of discrete (e.g., JPEG images) and continuous (e.g., H.263 video) media

There are 3 different methods to initiate a PSS session using different streaming tokens: either a SMIL presentation description, or an SDP session description, or an RTSP URL. A streaming token can get to a Device as a download from a server, or by super-distribution from other Devices, or by other means like user input of an RTSP URL via the keyboard.

Time-continuous protected media like audio and video tracks that are stored on a PSS server in the 3GP file format defined by 3GPP can either be downloaded by (progressive) download of the whole 3GP file, or streamed by extraction of protected media tracks from the 3GP file format and transport using real-time transport protocols. OMA has adopted the 3GP file format for protected packetised content as a special DCF, the Packetised DCF (PDCF) [DRMDCF-v2.1]. It should thus be understood that a 3GP file holding encrypted tracks as defined in [TS26.244] is automatically a valid OMA DRM PDCF [DRMDCF-v2.1].

Figure 25 gives an overview of the involved entities and data flows for DRM protection of 3GPP PSS sessions and media.

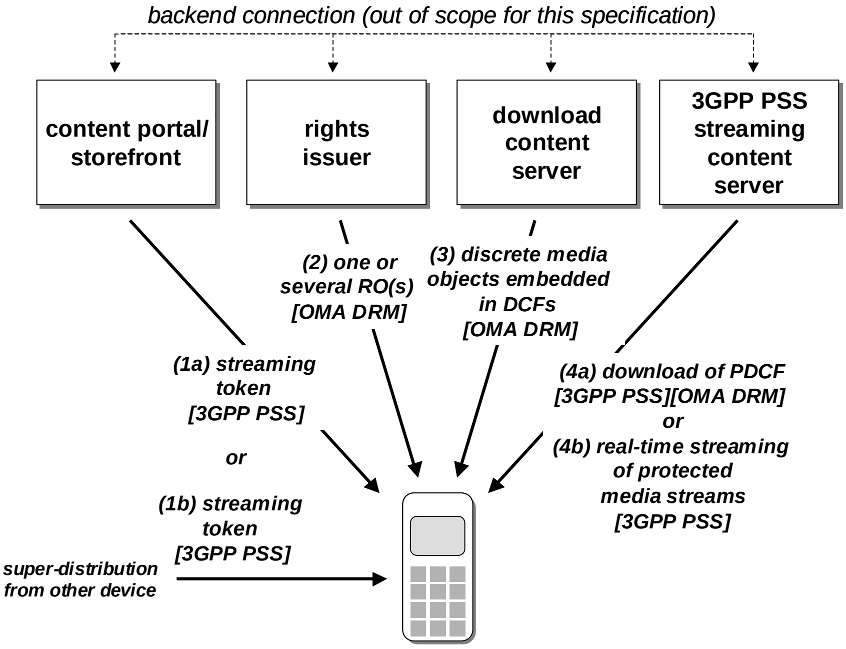
backend connection (out of scope for this specification)
content portal/ storefront
rights issuer
download content server
3GPP PSS streaming content server
(2) one or several RO(s) [OMA DRM]
(3) discrete media objects embedded in DCFs [OMA DRM]
(1a) streaming token [3GPP PSS]
or
(1b) streaming token [3GPP PSS]
(4a) download of PDCF [3GPP PSS][OMA DRM]
or
(4b) real-time streaming of protected media streams [3GPP PSS]
super-distribution from other device

Figure 25: Application of OMA DRM to the 3GPP Packet-Switched Streaming Service (Release 6).

(References in brackets indicate where the respective data format or protocol is specified)

For a protected PSS presentation, the content provider can confidentiality protect and integrity protect discrete media (images etc.) by embedding them into OMA DCFs. Further, he can confidentiality protect and integrity protect continuous media using the mechanisms defined by 3GPP, and storing them in a file in the 3GP file format [TS26.244], i.e., in a PDCF. The DCFs are stored on a content download server, the protected 3GP files = PDCFs on a 3GPP PSS server. Note that the PDCF can later be used for download or streaming of the included tracks/streams ((4a) or (4b) in Figure 25).

All information needed to generate ROs for the DCFs and PDCFs must be conveyed to the Rights Issuer; how this is done is outside the scope of this specification. This information includes the used content encryption keys for the discrete and continuous media, and usage rights/permissions.

The required steps to initiate, set up, receive, and render a protected 3GPP PSS session are then the following:

A streaming session is initiated via a streaming token, i.e. a SMIL presentation, SDP file, or RTSP URL [3GPP PSS]. The streaming token can arrive to the Device by download from a server/content portal/content storefront (see (1a) in Figure 25), or by super-distribution (see (1b) in Figure 25), by messaging (MMS), or by other means (e.g. an RTSP URL can be manually entered by the user). The streaming token can optionally be embedded into a DCF.

If the streaming token has been acquired directly from a server or portal, the server can initiate the delivery of one or several ROs to the Device that contain the keys and rights for the media referenced by the token (see (2) in Figure 25). In all other cases, the ROs for protected streams are requested during session setup to the streaming server, and the ROs for protected discrete objects after download of the respective DCFs, see (D)

When the user decides to start the PSS streaming session, she or he executes/launches the streaming token which is delivered to the streaming player. The streaming player evaluates the streaming token.

Depending on the type of streaming token, the following applies:

a. SMIL presentation: Referenced discrete objects are downloaded from the respective download servers (see (3) and (4a) in Figure 25). If ROs are not on the Device yet they can be acquired at this point, using the RI URL in the DCFs . Referenced streams are set up and started using PSS streaming protocols [3GPP PSS] (see (4b) in Figure 25). If ROs are not on the Device yet they can be acquired at this point, using the RI URL. Note: SMIL allows to download objects / start streams during a presentation. In this case it may be an implementation optimisation to fetch all ROs before starting the presentation.

b. RTSP URL: Referenced streams are set up and started using PSS streaming protocols [3GPP PSS] (see (4b) in Figure 25). If ROs are not on the Device yet they can be acquired at this point, using the RI URL.

c. SDP: Referenced streams are set up and started using PSS streaming protocols [3GPP PSS] (see (4b) in Figure 25). If ROs are not on the Device yet they can be acquired at this point, using the RI URL.

Discrete objects (DCFs), downloaded PSS content (PDCFs), and PSS streams are decrypted and rendered subject to the terms and permissions of the respective ROs.

The streaming token can be super-distributed to another Device. To be able to receive and render the referenced PSS media content, the receiving Device must acquire the respective RO(s).

D.1.2 DCF Packaging of Streaming Session Descriptors

The section describes an optional variation of the basic architecture and method for protection of streams using OMA DRM. In this variation, the streaming token / streaming session description is itself packaged into a DCF. This is illustrated in Figure 26.

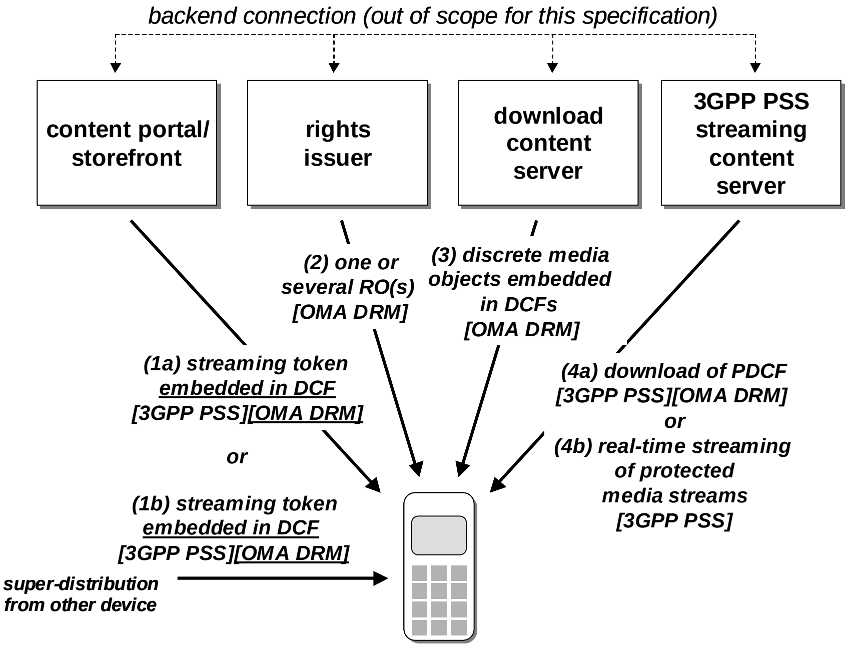
backend connection (out of scope for this specification)
content portal/ storefront
rights issuer
download content server
3GPP PSS streaming content server
(2) one or several RO(s) [OMA DRM]
(3) discrete media objects embedded in DCFs [OMA DRM]
(1a) streaming token embedded in DCF [3GPP PSS][OMA DRM]
or
(1b) streaming token embedded in DCF [3GPP PSS][OMA DRM]
super-distribution from other device
(4a) download of PDCF [3GPP PSS][OMA DRM]
or
(4b) real-time streaming of protected media streams [3GPP PSS]

Figure 26: Application of OMA DRM to the 3GPP Packet-Switched Streaming Service (Release 6) with streaming token packaged into DCF.

(Underlined text denotes differences to Figure 25.)

With this method, the typical steps to initiate, set up, receive, and render a protected 3GPP PSS session are similar as described in section D.1.1, with a few differences. The differences are outlined below.

(A) *Unchanged, see section D.1.1*.

(B) If the streaming token has been acquired directly from a server or portal, the server can initiate the delivery of one or several ROs to the Device that contain the keys and rights for the media referenced by the token (see (2) in Figure 26). Otherwise, the Device can use the RI URL in the streaming token DCF to request Rights Objects. If the RO or ROs delivered in response to this request contain the keys and rights for all media elements and streams being part of the PSS session associated with the token, no further RO requests are necessary.

(C) *Unchanged, see section D.1.1*.

(D) Depending on the type of streaming token, the following applies:

a. SMIL presentation: Referenced discrete objects are downloaded from the respective download servers (see (3) and (4a) in Figure 26). Referenced streams are set up and started using PSS streaming protocols [3GPP PSS] (see (4b) in Figure 26).

b. RTSP URL: Referenced streams are set up and started using PSS streaming protocols [3GPP PSS] (see (4b) in Figure 26). Please note that an RTSP URL per se cannot be packaged into a DCF, because there is no MIME type for RTSP URLs. However, a workaround is to package the RTSP URL into a helper file (e.g. a minimal SMIL file), and package the helper file into a DCF.

c. SDP: Referenced streams are set up and started using PSS streaming protocols [3GPP PSS] (see (4b) in Figure 26).

(E) *Unchanged, see section D.1.1*.

(F) *Unchanged, see section D.1.1*.

A difference using the optional method is the point in time when ROs are requested/acquired: it is always (including the super-distribution case) possible to request rights when the DCF containing the streaming token is available on the Device, and before streaming of content is initiated. If the RO (or ROs) delivered in response contain rights and keys for all media objects and streams used in the respective PSS presentation, no further RO requests are necessary.

Also, the RI can associate permissions or constraints with the streaming token, in addition to constraints on the referenced media objects or streams. For example, for datetime based restrictions on streams, the same restriction could be imposed on the token. If the user tries to use the streaming token after expiry, this is then recognised when the token is executed, and before any communication with the streaming server is set up.

All DCF-associated functionality is applicable to a streaming token packaged into a DCF (e.g., integrity protection of DCF, preview rights URL, etc.).

The described optional method of packaging streaming tokens into DCFs has no implications on the security or protection of the referenced media objects and streams.

Appendix E. Certificate Profiles and Requirements

E.1 DRM Agent Certificates

The profile for DRM Agent certificates follows the profile for "User Certificates for Authentication" in [CertProf] with the following modifications:

Version	3
Signature	MUST be RSA with SHA-1
Serial Number	MUST be less than, or equal to, 20 bytes in length
Issuer Name	MUST be present and MUST use a subset of the following naming attributes from [CertProf] – countryName, organisationName, organisationalUnitName, commonName, and stateOrProvinceName.
Subject Name	MUST be present and MUST use a subset of the following naming attributes from [CertProf] – countryName, organisationName, organisationalUnitName, commonName, and serialNumber. The structure and contents of a Device subject name shall be as follows: [countryName=<Country of manufacturer>] [organisationName=<Manufacturer company name>] [organisationalUnitName=<Manufacturing location>] [commonName=<Model name>] serialNumber=<Unique identifier for Device, as assigned by the Certificate Issuer. Does not have to be the same as the IMEI> The serialNumber attribute MUST be present. The countryName, organisationName, organisationalUnitName, and commonName may be present. Other attributes are not allowed and must not be included. For all naming attributes of type DirectoryString, the PrintableString or the UTF8String choice must be used. Note that the maximum length (in octets) for values of these attributes is as follows: countryName – 2 (country code in accordance with ISO/IEC 3166), organisationName, organisationalUnitName, commonName, and serialNumber – 64. Example: C="US";O="DRM Devices 'R Us"; CN="DRM Device Mark V"; SN="1234567890"
Extensions	The extKeyUsage extension SHALL be present, and contain (at least) the **`oma-kp-drmAgent`** key purpose object identifier: `oma-kp-drmAgent OBJECT IDENTIFIER ::= {oma-kp 2}` The oma-kp object identifier is defined as follows:

	`oma-kp OBJECT IDENTIFIER ::= {oma 1}`
	`oma OBJECT IDENTIFIER ::= {joint-iso-itu-t(2)`
	`identified-organisations(23)`
	`wap(43) oma(6)}`
	CAs are recommended to set this extension to critical.
	If CAs include the keyUsage extension (recommended), then both the digitalSignature bit and the keyEncipherment bit must be set, if the corresponding private key is to be used both for authentication and decryption. Otherwise only the applicable bit shall be set. When present, this extension shall be set to critical.
	CAs may include the certificatePolicy extension, indicating the policy the certificate has been issued under, and possibly containing a URI identifying a source of more information about the policy.
	CAs are recommended to not include any other extensions, but may, for compliance with [RFC3280], include the authorityKeyIdentifier extension. CAs may also include the authorityInfoAccess extension from [RFC3280] for OCSP responder navigation purposes, and the cRLDistributionPoints extension to identify how CRL information is obtained.
	CAs MUST NOT include any other critical extensions.

RI implementations MUST meet all requirements on entities processing user certificates defined in [CertProf]. In addition, RIs:

MUST be able to process DRM Agent certificates with serial numbers up to 20 bytes long;

MUST recognize and require the presence of the **`oma-kp-drmAgent`** object identifier defined above in the extKeyUsage extension in DRM Agent certificates; and

MUST support the cRLDistributionPoints extension.

E.2 Rights Issuer Certificates

The profile for RI certificates follows the profile for "X.509-compliant server certificate" in [CertProf] with the following modifications:

Signature	MUST be RSA with SHA-1
Serial Number	MUST be less than, or equal to, 20 bytes in length
Issuer Name	MUST be present and MUST use a subset of the following naming attributes from [CertProf] – countryName, organisationName, organisationalUnitName, commonName, and stateOrProvinceName.
Subject Name	MUST be present and MUST use a subset of the following naming attributes from [CertProf] – countryName, stateOrProvinceName, localityName, organisationName, organisationalUnitName, and commonName. The structure and contents of a Rights Issuer subject name shall be as follows: countryName=<Country of operation>

	[stateOrProvinceName=<State/Province>] [localityName=<City>] organisationName=<RI company name> [organisationalUnitName=<RI subsidiary/location>] commonName=<RI company name> "OMA Rights Issuer" [<serNo>] (For the commonName attribute, the <serNo> string is specified when a given organisation has several RIs.) The countryName, organisationName, and commonName naming attributes must be present. The stateOrProvinceName, localityName, and/or organisationalUnitName naming attributes may be present. Other attributes are not allowed and must not be included. For all naming attributes of type DirectoryString, the PrintableString or the UTF8String choice must be used. Note that the maximum length (in octets) for values of these attributes is as follows: countryName – 2, stateOrProvinceName and localityName – 128, organisationName, organisationalUnitName, and commonName – 64. Example: C="US";O="ROs for everyone"; CN="ROs for everyone OMA Rights Issuer"
Extensions	The extKeyUsage extension shall be present, and contain (at least) the **oma-kp-rightsIssuer** key purpose object identifier: `oma-kp-rightsIssuer OBJECT IDENTIFIER ::= {oma-kp 1}` CAs are recommended to set this extension to critical. If the keyUsage extension is present (recommended), then the digitalSignature bit shall be set. When present, this extension shall be set to critical. CAs may include the certificatePolicy extension, indicating the policy the certificate has been issued under, and possibly containing a URI identifying a source of more information about the policy. CAs are recommended to not include any other extensions, but may, for compliance with [RFC3280], include the authorityKeyIdentifier extension. CAs may also include the authorityInfoAccess extension from [RFC3280] for OCSP responder navigation purposes. CAs MUST NOT include any other critical extensions.

DRM Agents processing Rights Issuer certificates MUST meet the requirements on clients processing "X.509-compliant server certificates" defined in [CertProf]. In addition, DRM Agents:

MUST be able to process RI certificates up to 1500 bytes long;

MUST be able to process RI certificates with serial numbers 20 bytes long; and

MUST recognize and require the presence of the **oma-kp-rightsIssuer** object identifier defined above in the extKeyUsage extension in RI certificates.

E.3 CA Certificates

The profile for OMA DRM CA certificates follows the profile for "Authority Certificates" in [CertProf] with the following modifications:

Signature	MUST be RSA with SHA-1
Serial Number	MUST be less than, or equal to, 20 bytes in length

RIs and DRM Agents MUST meet the requirements on relying parties defined in [CertProf]. Note that this implies, among other things, a requirement on RIs and DRM Agents to also recognize the basicConstraints and the subjectKeyIdentifier extensions. In addition, DRM Agents:

MUST be able to process authority certificates up to 1500 bytes long; and

MUST be able to process authority certificates with serial numbers 20 bytes long.

E.4 OCSP Responder Certificates

The profile for OCSP responder certificates in [OCSP-MP] applies. RIs and DRM Agents MUST meet the requirements on "Authority Certificate" relying parties defined in [CertProf]. In addition, RIs and DRM Agents:

MUST be able to process OCSP responder certificates up to 1500 bytes long;

MUST be able to process OCSP responder certificates with serial numbers 20 bytes long;

MUST recognize the extKeyUsage extension and its **id-kp-OCSPSigning** object identifier (i.e. support OCSP responder delegation): and

MUST recognize the **id-pkix-ocsp-noCheck** certificate extension defined in [OCSP], indicating that the certificate is non-revocable (i.e. no certificate status information will be provided for it).

E.5 User Certificates for Authentication

The profile specified in [CertProf] MUST be used. If a Device supports WIM and binding to the WIM, then, note that this implies a requirement on DRM Agents to also recognize the keyUsage, extKeyUsage, certificatePolicies, subjectAltName, and basicConstraints extensions.

Appendix F. Interactions between the DRM Agent and the WIM(Informative)

F.1 WIM Operations in Exercising “permission” to bind Rights Objects to the User Identity

This appendix describes messages sent between the DRM agent and the WIM when the DRM agent needs to verify the presence of a WIM bound to an RO. The message flow between the DRM Agent and the WIM is described at a functional level, using service primitives.

The preliminary exchanges based on device control and verification related primitives c.f., [WIM] are intentionally omitted from this flowchart but MAY be required.

The DRM Agent must set the WIM_GENERIC_RSA Security Environment to perform the signature operations.

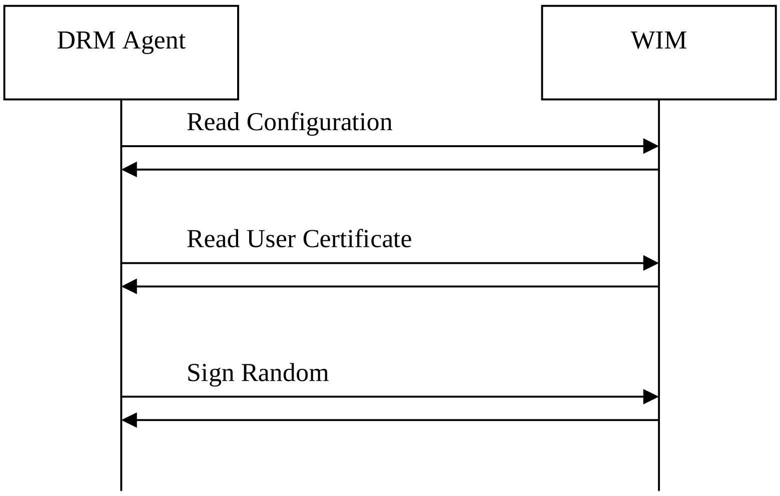

Figure 27: DRM Agent and WIM Interaction

Read configuration

Before starting the procedure, the DRM Agent needs to know which algorithms the WIM supports and information on keys and certificates stored in the WIM.

To read the configuration the DRM Agent uses data access primitives: WIM-OpenFile, WIM-ReadBinary etc.

Read user certificate

The DRM Agent may read the user certificate stored in the WIM and identified by PKCS_iD.

To read the user certificate the DRM Agent uses data access primitives: WIM-OpenFile, WIM-ReadBinary etc.

Sign random

The WIM has to sign the challenge number sent by the DRM Agent and return the signature. The DRM Agent may successfully verify the signature prior to exercise the permission.

To get the signature the DRM Agent uses the WIM-ComputeDigitalSignature primitive. The primitive returns the signature.

F.2 PIN Management

Said user private key is protected by a PIN-G (Global PIN), thus the procedure may require PIN-G verification i.e., the DRM Agent may have to send the WIM-Perform-Verification primitive one time per WIM session. Once PIN-G right is granted, the procedure does not require PIN-G verification anymore for the current WIM session.

Note: in case the WIM application is present on a UICC smart card platform [UICC] together with a USIM [3GPP TS 31.102] application, the WIM PIN-G can be mapped on the USIM PIN.

Appendix G. Static Conformance Requirements (Normative)

The notation used in this appendix is specified in [IOPPROC].

G.1 Client Conformance Requirements

The table below enumerates the client conformance requirements on all Devices – Connected, as well as Unconnected Devices. The Enabler Release Definition for DRM V2.1 [DRMERELD-v2.1] defines the mandatory features to be supported by the Connected and Unconnected Devices. For further information, please see section 8 of [DRMERELD-v2.1].

Item	Function	Reference	Status	Requirements
DRM-CLI-CMN-001	ROAP Schema parsing and processing support.	5.3	M	
DRM-CLI-CMN-002	General XML Schema Requirements	5.3.2	M	
DRM-CLI-CMN-003	Nonce values in ROAP messages	5.3.12	M	
DRM-CLI-CMN-004	Processing and responding to status codes during ROAP protocol runs	5.3.6,5.4.2	M	
DRM-CLI-CMN-005	ROAP Trigger parsing and processing	5.2.1	M	
DRM-CLI-CMN-006	ProtectedRO support	5.3.9,5.3.10	M	
DRM-CLI-CMN-007	XML Canonicalisation	5.3.10,5.4	M	
DRM-CLI-CMN-008	4-pass ROAP-Registration protocol	5.4.2	M	
DRM-CLI-CMN-009	ROAP Extensions	5.4.2,5.4.3, 5.4.5	O	
DRM-CLI-CMN-010	Hash Algorithms: SHA-1 and associated URI	5.4.2.1.1	M	
DRM-CLI-CMN-011	MAC Algorithms: HMAC-SHA-1 and associated URI	5.4.2.1.1	M	
DRM-CLI-CMN-012	Signature Algorithms: RSA-PSS-Default and associated URI	5.4.2.1.1	M	
DRM-CLI-CMN-013	Key Transport Algorithms: RSAES-KEM-KDF2-KW-AES128 and associated URI	5.4.2.1.1	M	
DRM-CLI-CMN-014	Key Wrap Algorithms: AES-WRAP and associated URI	5.4.2.1.1	M	
DRM-CLI-CMN-015	Domains Functionality	5.1.7,5.1.8, 5.4.5,7.2.3, 7.3,8	O	0,0,0,0,0,0, AND 0, 0 OR 0, 0
DRM-CLI-CMN-016	Hash Chains for Domain Key Management	5.4.5.1.1,7.3,8.8.1	O	
DRM-CLI-CMN-017	DRM Agent Certificates	E.1	M	
DRM-CLI-CMN-018	User Certificates for WIM Binding	E.5	O	
DRM-CLI-CMN-019	RI Certificate Processing and Certificate Chain Validation	5.4.2.4,5.4.4.2,5.4.5.2, 6.2	M	
DRM-CLI-CMN-020	RI Signature Validation	5.4.2.4,5.4.4.2,5.4.5.2	M	
DRM-CLI-CMN-021	OCSP Response Validation	5.4.2.4,5.4.4.2,5.4.5.2, 6.2,6.3	M	OCSP-C-006, OCSP-C-007, OCSP-C-009, OCSP-C-011, OCSP-C-012, OCSP-C-013, OCSP-C-015,

				OCSP-C-016, OCSP-C-017, OCSP-C-019, OCSP-C-020, OCSP-C-021, OCSP-C-022, OCSP-C-022a, OCSP-C-022b, OCSP-C-022c, OCSP-C-023, OCSP-C-024, OCSP-C-028
DRM-CLI-CMN-022	IMSI Binding	19.1	O	
DRM-CLI-CMN-023	WIM Binding	19.2	O	0
DRM-CLI-CMN-024	Transaction Tracking	15.3, 5.4.4.1, 5.4.4.2.1	O	
DRM-CLI-CMN-025	User Consent for ROAP Triggers and associated processing	5.2.1	M	
DRM-CLI-CMN-026	User Consent for Silent and Preview Headers	5.2.2	M	
DRM-CLI-CMN-027	RI Certificate Caching	5.4.2.1.1	O	
DRM-CLI-CMN-028	RI Certificate Verification data storage in the RI Context	5.4.2.4.1	O	
DRM-CLI-CMN-029	Replay Protection for Stateful Rights Objects	10.4,5.3.10	M	
DRM-CLI-CMN-030	Maintaining state information for Stateful Rights Objects	10.4.1	M	
DRM-CLI-CMN-031	Domain Name Whitelists	5.4.2.4.1	M	
DRM-CLI-CMN-032	Multiple Domain Contexts	8.2	O	
DRM-CLI-CMN-033	Domain Context	5.4.5.2.1,8.2	O	
DRM-CLI-CMN-034	Domain Context Expiry processing	5.4.5.2.1	O	
DRM-CLI-CMN-035	Installing Domain ROs	8.7.2.1, 8.7,5.4.5.2	O	
DRM-CLI-CMN-036	Multiple RI Contexts	5.4.2.4.1	M	
DRM-CLI-CMN-037	RI Context	5.4.2.4.1	M	
DRM-CLI-CMN-038	Use of riID as identifiers for RI Contexts stored in the Device	5.4.2.4.1,5.3.9,5.2.1	M	
DRM-CLI-CMN-039	RI Context Expiry processing	5.4.2.4.1	M	
DRM-CLI-CMN-040	DCF Hash verification; usage in ROAP	5.4.4.1.1	O	
DRM-CLI-CMN-041	Device RO Processing	10.3.1	M	
DRM-CLI-CMN-042	Domain RO Processing	8.7	O	
DRM-CLI-CMN-043	MIME Types for ROAP PDU, Trigger, ProtectedRO, and Rights Objects	5.3.9,13.2	M	
DRM-CLI-CMN-044	Exporting to other DRMs and Protected Links	16	O	
DRM-CLI-CMN-045	Super Distribution of the DCF	15	O	
DRM-CLI-CMN-046	Super Distribution of the ContentURL	15	O	
DRM-CLI-CMN-047	Parent Rights Object	10.5	M	
DRM-CLI-CMN-048	Off-device storage of content and Rights Objects	10.6	O	
DRM-CLI-CMN-049	Capability signaling to Content Issuers and Rights Issuers	11	M	

DRM-CLI-CMN-050	Processing Content Objects, Rights Objects and ROAP Triggers received via WAP PUSH	14.4	M	
DRM-CLI-CMN-051	DCF Integrity protection after the DCFs are downloaded to the Device	15.4	M	
DRM-CLI-CMN-052	Backwards Compatibility to OMA DRM v1	Appendix C	M	
DRM-CLI-CMN-053	DRM Time	6.3,5.4	O	0
DRM-CLI-CMN-054	DRM Time Synchronisation	6.3,5.4	O	
DRM-CLI-CMN-055	Connectivity for Unconnected Devices via ROAP over OBEX	14.6	O	0
DRM-CLI-CMN-056	2-pass ROAP-ROAcquisition protocol	5.4.3	O	
DRM-CLI-CMN-057	1-pass ROAP-ROAcquisition protocol	5.4.4.2.1	O	
DRM-CLI-CMN-058	2-pass ROAP-JoinDomain protocol	5.4.5.1	O	
DRM-CLI-CMN-059	2-pass ROAP-LeaveDomain protocol	5.4.5.3	O	
DRM-CLI-CMN-060	HTTP Transport Mapping	14.2	O	
DRM-CLI-CMN-061	Capability Signalling	13	O	
DRM-CLI-CMN-062	Silent and Preview header processing in DCFs	5.2.2	O	
DRM-CLI-CMN-063	Download OTA support for delivering Content , ROAP Triggers, and Rights Objects	14.3	O	
DRM-CLI-CMN-064	Utilize the connectivity provided by the Connected Device to conduct ROAP protocols	18	O	
DRM-CLI-CMN-065	ROAP-OBEX Server	18,14.6	O	
DRM-CLI-CMN-066	2-pass ROAP JoinDomain protocol	5.4.5.1	O	
DRM-CLI-CMN-067	2-pass ROAP LeaveDomain protocol	5.4.5.3	O	
DRM-CLI-CMN-068	Metering	5.4.6, 11	O	
DRM-CLI-CMN-069	2-pass ROAP RO Upload protocol	5.4.7, 12	O	
DRM-CLI-CMN-070	WBXML ROAP Trigger	16.2	O	
DRM-CLI-CMN-072	4-pass confirmed ROAP-ROAcquisition Protocol	5.1.4, 5.4.4.3, 5.4.4.4	O	
DRM-CLI-CMN-072	3-pass confirmed ROAP-ROResponse Protocol	5.1.6, 5.4.4.3, 5.4.4.4	O	
DRM-CLI-CMN-073	2-pass Identification Protocol	5.1.2	O	

G.2 Server Conformance Requirements

Item	Function	Reference	Status	Requirements
DRM-SERVER-001	ROAP schema parsing and message processing	5.3	M	
DRM-SERVER-002	General XML Schema Requirements	5.3.2	M	
DRM-SERVER-003	Nonce values in ROAP messages	5.3.12	M	
DRM-SERVER-004	Indicating the status parameter in the runs of the ROAP protocols as defined	5.3.6,5.4.2	M	
DRM-SERVER-005	XML Canonicalisation	5.3.10,5.4	M	
DRM-SERVER-006	RI Certificates	E.2	M	
DRM-SERVER-007	DRM Agent Certificate processing and Certificate Chain Validation	5.4.2.3.1	M	
DRM-SERVER-008	Unique riID in ROAP Protocols.	5.4	M	
DRM-SERVER-009	Support for OCSP Requests; including nonce extensions.	5.4.2.4.1	M	OCSP-C-001, OCSP-C-002, OCSP-C-004,

				OCSP-C-006, OCSP-C-007, OCSP-C-025, OCSP-C-027, OCSP-C-031 OCSP-C-033, OCSP-C-034, OCSP-C-035, OCSP-C-037 See Note [2]
DRM-SERVER-010	Providing the most recent OCSP Response to Devices in ROAP protocol runs	5.4.2.4.1	O	
DRM-SERVER-011	ROAP Trigger support and initiating the ROAP protocol using ROAP Triggers	5.2.1	M	
DRM-SERVER-012	Domain ID element in ROAP Triggers	5.2.1	O	
DRM-SERVER-013	More than one roID elements in a roAcquisition trigger	5.2.1	O	
DRM-SERVER-014	Use of MAC in leaveDomain ROAP Trigger	5.2.1	M	
DRM-SERVER-015	4-pass ROAP-Registration Protocol	5.4.2	M	
DRM-SERVER-016	2-pass ROAP-ROAcquisition Protocol	5.4.3	M	
DRM-SERVER-017	1-pass ROAP-ROResponse Protocol	5.4.4.2.1	M	
DRM-SERVER-018	2-pass ROAP-JoinDomain Protocol	5.4.5.1	M	
DRM-SERVER-019	2-pass ROAP-LeaveDomain Protocol	5.4.5.3	M	
DRM-SERVER-020	Hash Chain support for Domain Key Generation	8.8.1	O	
DRM-SERVER-021	ProtectedRO support	5.3.9	M	
DRM-SERVER-022	Signature on Domain RO	5.4.4.2.1,5.3.10	M	
DRM-SERVER-023	Signature on Device RO	5.3.10,5.4.4.2.1	O	
DRM-SERVER-024	domainRO and riURL attributes in ProtectedRO for Domain ROs	5.3.10	M	
DRM-SERVER-025	Hash Algorithms: SHA-1 and associated URI	5.4.2.1.1	M	
DRM-SERVER-026	MAC Algorithms: HMAC-SHA-1 and associated URI	5.4.2.1.1	M	
DRM-SERVER-027	Signature Algorithms: RSA-PSS-Default and associated URI	5.4.2.1.1	M	
DRM-SERVER-028	Key Transport Algorithms: RSAES-KEM-KDF2-KW-AES128 and associated URI	5.4.2.1.1	M	
DRM-SERVER-029	Key Wrap Algorithms: AES-WRAP and associated URI	5.4.2.1.1	M	
DRM-SERVER-030	Unique identifier for Rights Issuers	5.3.10	M	
DRM-SERVER-031	Parent Rights Object	10.5	M	
DRM-SERVER-032	Issuer Responsibilities	13.4	M	
DRM-SERVER-033	Download OTA support for delivering Content,	14.3	O	

[2] Note: The RI is used primarily as a proxy between the DRM agent and the OCSP responder and thus does not necessarily need to process the OCSP response. However, to minimize client side processing and to reduce bandwidth consumption, this specification highly recommends that Rights Issuers do as much processing and validation of OCSP responses it receives from the responder as possible before sending them to the DRM agent and thus also support OCSP-C-009, OCSP-C-011, OCSP-C-012, OCSP-C-013, OCSP-C-015, OCSP-C-016, OCSP-C-017, OCSP-C-019, OCSP-C-021, OCSP-C-022b, OCSP-C-022c, OCSP-C-029, OCSP-C-030.

	ROAP Triggers, and Rights Objects			
DRM-SERVER-034	Use of WAP PUSH to deliver Content, ROAP Triggers, and Rights Objects	14.4	M	
DRM-SERVER-035	Transaction Tracking	15.3, 5.4.4.1, 5.4.4.2.1	M	
DRM-SERVER-036	Metering	5.4.6,11	M	
DRM-SERVER-037	2-pass ROAP RO Upload protocol	5.4.7, 12	O	
DRM-SERVER-038	WBXML ROAP Trigger	16.2	O	
DRM-SERVER-039	4-pass confirmed ROAP-ROAcquisition Protocol	5.1.4, 5.4.4.3, 5.4.4.4	M	
DRM-SERVER-040	3-pass confirmed ROAP-ROResponse Protocol	5.1.6, 5.4.4.3, 5.4.4.4	M	
DRM-SERVER-041	2-pass Identification Protocol	5.1.2	O	

Appendix H. Examples (Informative)

H.1 ROAP Examples

All examples are syntactically correct. Signature, MAC, cipher and digest values are fictitious however.

According to 5.3.3, these messages must be canonicalised before being sent by the RI or the DRM Agent.

H.1.1 Device hello

```
<roap:deviceHello
  xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <version>1.1</version>
    <deviceID>
        <keyIdentifier xsi:type="roap:X509SPKIHash">
            <hash>vXENc+Um/9/NvmYKiHDLaErK0gk=</hash>
        </keyIdentifier>
    </deviceID>
</roap:deviceHello>
```

H.1.2 RI Hello

```
<roap:riHello
  xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  status="Success" sessionId="433211">
    <selectedVersion>1.1</selectedVersion>
    <riID>
        <keyIdentifier xsi:type="roap:X509SPKIHash">
            <hash>aXENc+Um/9/NvmYKiHDLaErK0fk=</hash>
        </keyIdentifier>
    </riID>
    <riNonce>dsaiuiure9sdwerfqwer</riNonce>
    <trustedAuthorities>
        <keyIdentifier xsi:type="roap:X509SPKIHash">
            <hash>bew3e332oihde9dwiHDLaErK0fk=</hash>
        </keyIdentifier>
        <keyIdentifier xsi:type="roap:X509SPKIHash">
            <hash>3lkpoi9fceoiuoift45epokifc0poiss</hash>
        </keyIdentifier>
    </trustedAuthorities>
    <extensions>
        <extension xsi:type="roap:CertificateCaching"/>
    </extensions>
</roap:riHello>
```

H.1.3 Registration Request

```
<roap:registrationRequest
  xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  sessionId="433211">
    <nonce>32efd34de39sdwefqwer</nonce>
    <time>2004-03-17T14:20:00Z</time>
    <certificateChain>
        <certificate>miib123121234567</certificate>
        <certificate>miib234124312431</certificate>
  </certificateChain>
  <trustedAuthorities>
    <keyIdentifier xsi:type="roap:X509SPKIHash">
      <hash>432098mhj987fdlkj98lkj098lkjr409</hash>
    </keyIdentifier>
    <keyIdentifier xsi:type="roap:X509SPKIHash">
      <hash>432098ewew5jy6532fewfew4f43f3409</hash>
    </keyIdentifier>
  </trustedAuthorities>
  <signature>321ue3ue3ue10ue2109ue1ueoidwoijdwe309u09ueqijdwqijdwq09uwqwqi009</signature>
</roap:registrationRequest>
```

H.1.4 Registration Response

```
<roap:registrationResponse
 xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 status="Success" sessionId="433211" >
  <riURL>http://ri.example.com/roap.cgi</riURL>
  <certificateChain>
    <certificate>miib123121234567</certificate>
    <certificate>miib234124312431</certificate>
  </certificateChain>
  <ocspResponse>fdow9rw0feijfdsojr3w09u3wijfdslkj4sd</ocspResponse>
  <signature>321ue3ue3ue10ue2109ue1ueoidwoijdwe309u09ueqijdwqijdwq09uwqwqi009</signature>
</roap:registrationResponse>
```

H.1.5 RO Request

The request is for a Device RO.

```
<roap:roRequest
 xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <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>
  <time>2004-03-17T14:20:00Z</time>
  <roInfo>
    <roID>n8yu98hy0e2109eu09ewf09u</roID>
  </roInfo>
  <certificateChain>
    <certificate>miib123121234567</certificate>
    <certificate>miib234124312431</certificate>
  </certificateChain>
  <signature>321ue3ue3ue10ue2109ue1ueoidwoijdwe309u09ueqijdwqijdwq09uwqwqi009</signature>
</roap:roRequest>
```

H.1.6 RO Request with dcfHash

The request is for a Device RO using dcfHash and Device RO referring to 2 dcf Files. Note that the roIDs are identical and the hash values different.

```
<roap:roRequest
  xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <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>
    <time>2004-03-17T14:20:00Z</time>
    <roInfo>
        <roID>n8yu98hy0e2109eu09ewf09u</roID>
        <dcfHash algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
        <hash>AFGK+7/diuUSH7hdjkaq==</hash>
        </dcfHash>
        <roID>n8yu98hy0e2109eu09ewf09u</roID>
        <dcfHash algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
        <hash>12JDik684d/kdjhap9KDZ==</hash>
        </dcfHash>
    </roInfo>
    <certificateChain>
        <certificate>miib123121234567</certificate>
        <certificate>miib234124312431</certificate>
    </certificateChain>
    <signature>321ue3ue3ue10ue2109ue1ueoidwoijdwe309u09ueqijdwqijdwq09uwqwqi009</signature>
</roap:roRequest>
```

H.1.7 RO Response

The response is a Rights Object intended for the recipient only. Note that the response indicates that the Rights Object is stateful.

```
<roap:roResponse
  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"
  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>
    <roap:protectedRO>
        <roap:ro id="n8yu98hy0e2109eu09ewf09u" stateful="true" version="1.1">
```

```
         <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>2.1</o-dd:version>
               <o-dd:uid>n8yu98hy0e2109eu09ewf09u</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:constraint>
                          <o-dd:count>1</o-dd:count>
                      </o-ex:constraint>
                 </o-dd:play>
               </o-ex:permission>
            </o-ex:agreement>
         </rights>
         <encKey Id="K_MAC_and_K_REK">
            <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>231jks231dkdwkj3jk321kj321j321kj423j342h213j321jh321jh2134jhk3211fdslfdsopfespj
oefwopjsfdpojvct4w925342a</xenc:CipherValue>
            </xenc:CipherData>
         </encKey>
      </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>
  <ocspResponse>miibewqoidpoidsa</ocspResponse>
  <extensions>
    <extension xsi:type="roap:TransactionIdentifier">
      <contentID>ContentID</contentID>
              <id>09321093209-2121</id>
    </extension>
  </extensions>
  <signature>d93e5fue3susdskjhkjedkjrewh53209efoihfdse10ue2109ue1</signature>
</roap:roResponse>
```

H.1.8 RO Confirm Request

This request confirms the installation of the ROResponse in the previous section..

```
<roap:roConfirmRequest
 xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 sessionId="433213">
  <riID>
    <keyIdentifier xsi:type="roap:X509SPKIHash">
      <hash>aXENc+Um/9/NvmYKiHDLaErK0fk=</hash>
    </keyIdentifier>
  </riID>
 <deviceID>
    <keyIdentifier xsi:type="roap:X509SPKIHash">
      <hash>vXENc+Um/9/NvmYKiHDLaErK0gk=</hash>
    </keyIdentifier>
  </deviceID>
  <nonce>32efd34sad78fqwer</nonce>
  <time>2004-03-17T14:30:00Z</time>
  <roConfirmInfo>
    <roID>n8yu98hy0e2109eu09ewf09u</roID>
    <status>true </status>
  </roConfirmInfo>
  <certificateChain>
    <certificate>miib123121234567</certificate>
    <certificate>miib234124312431</certificate>
  </certificateChain>
  <signature>d93e5fue8susksdnflsdkjrewh55832efoihfdse14ue3485nb</signature>
</roap:roConfirmRequest>
```

H.1.9 RO Confirm Response

```
<roap:roConfirmResponse xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  sessionId="433213" status="Success">
    <nonce>32efd34sad78fqwer</nonce>
    <riID>
       <keyIdentifier xsi:type="roap:X509SPKIHash">
           <hash>aXENc+Um/9/NvmYKiHDLaErK0fk=</hash>
       </keyIdentifier>
    </riID>
    <signature>
       d93e5fue8susksdnflsdkjrewh55832efoihfdse14ue3485nb
    </signature>
</roap:roConfirmResponse>
```

H.1.10 Domain RO

The Domain RO may be sent separately, as here, or within a ROAP-ROResponse.

```
<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="1.1"
      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>2.1</o-dd:version>
        <o-dd:uid>n8yu98hy0e2109eu09ewf09u</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: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>slo5hb+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>
 </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>
```

H.1.11 Join Domain Request

```
<roap:joinDomainRequest
 xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <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>
  <time>2004-03-17T14:30:00Z</time>
  <domainID>Domain-XYZ-001</domainID>
  <certificateChain>
    <certificate>miib123121234567</certificate>
    <certificate>miib234124312431</certificate>
  </certificateChain>
  <signature>321ue3ue3ue10ue2109ue1ueoidwoijdwe309u09ueqijdwqijdwq09uwqwqi009</signature>
</roap:joinDomainRequest>
```

H.1.12 Join Domain Response

```
<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>231jks231dkdwkj3jk321kj321j321kj423j342h213j321jh321jh2134jhk3211fdslfdsopfespj
oefwopjsfdpojvct4w925342a</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>
  <signature>d93e5fue3ue10ue2109ue1ueoidwoijdwe309u09ueqijdwqijdwq09uwqwqi009</signature>
</roap:joinDomainResponse>
```

H.1.13 Leave Domain Request

```
<roap:leaveDomainRequest
 xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 triggerNonce="sdfknjvfda438790fdjkl4rq">
  <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>
  <time>2004-03-17T19:20:00Z</time>
  <domainID>Domain-XYZ-001</domainID>
  <certificateChain>
    <certificate>miib123121234567</certificate>
    <certificate>miib234124312431</certificate>
  </certificateChain>
  <signature>321ue3ue3ue10ue2109ue1ueoidwoijdwe309u09ueqijdwqijdwq09uwqwqi009</signature>
</roap:leaveDomainRequest>
```

H.1.14 Leave Domain Response

```
<roap:leaveDomainResponse
 xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 status="Success">
  <nonce>32efd34de39sdwefqwer</nonce>
  <domainID>Domain-XYZ-001</domainID>
</roap:leaveDomainResponse>
```

H.1.15 MeteringReportSubmit

```
<roap:meteringReportSubmit
```

```
 xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#
 xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
 triggerNonce="sdfknjvfda438790fdjkl4rq">
  <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>
  <time>2004-03-17T19:20:00Z</time>
  <meteringReport>
    <encryptedMeteringReport>
      <xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Content">
        <xenc:EncryptionMethod Algorithm=" http://www.w3c.org/2001/04/xmlenc#aes128-cbc"/ >
        <ds:KeyInfo>
          <ds:RetrievalMethod URI="#K_MEK_and_K_MAC"/>
        </ds:KeyInfo>
        <xenc:CipherData>
          <xenc:CipherValue>A23B45C56</CipherValue>
        </xenc:CipherData>
      </xenc:EncryptedData>
    </encryptedMeteringReport>
    <encKey Id="K_MEK_and_K_MAC">
      <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>F98E76D54</xenc:CipherValue>
        </xenc:CipherData>
    </encKey>
    <mac>ewqrewoewfewohffohr3209832r3</mac>
  </meteringReport>
  <certificateChain>
    <certificate>miib123121234567</certificate>
    <certificate>miib234124312431</certificate>
  </certificateChain>
  <signature>321ue3ue3ue10ue2109ue1ueoidwoijdwe309u09ueqijdwqijdwq09uwqwqi009</signature>
</roap:meteringReportSubmit>
```

H.1.16 MeteringReportResponse

```
<roap:meteringReportResponse
 xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
 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>
  <certificateChain>
   <certificate>miib223121234567</certificate>
   <certificate>miib834124312431</certificate>
  </certificateChain>
  <signature>a56je3ue3ue10ue2109ue1ueoidwoijdwe309u09ueqijdwqijdwq09uwqwqi009</signature>
</roap:meteringReportResponse>
```

H.1.17 RO Upload Request

```
<roap:roUploadRequest
    xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
    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">
    <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>
  <time>2006-03-17T14:20:00Z</time>
  <roInfo>
   <roID>n8yu98hy0e2109eu09ewf09u</roID>
   <stateInfo o-ex:id="C-1">
     <o-dd:count>98</o-dd:count>
   </stateInfo >
   <stateInfo o-ex:id="C-2">
      <o-dd:count>8</o-dd:count>
      <o-dd:datetime>
        <o-dd:end>2005-10-21T00:00:00Z</o-dd:end>
       </o-dd:datetime>
   </stateInfo>
  </roInfo>
  <signature>821ue3ue3ue10ue2109ue1ueoidwoijdwe309u09ueqijdwqijdwq09uwqwqi009</signature>
</roap:roUploadRequest>
```

H.1.18 RO Upload Response

```
<roap:roUploadResponse
  xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
  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>
  <certificateChain>
    <certificate>miib223121234567</certificate>
    <certificate>miib834124312431</certificate>
   </certificateChain>
   <signature>321ue3ue3ue10ue2109ue1ueoidwoijdwe309u09ueqijdwqijdwq09uwqwqi009</signature>
</roap:roUploadResponse>
```

H.1.19 Identification Request Trigger

```
<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="1.1”>
  <extendedTrigger id="de32r23r3" type="identificationRequest">
    <riID>
      <keyIdentifier xsi:type="roap:X509SPKIHash">
        <hash>aXENc+Um/9/NvmYKiHDLaErK0fk=</hash>
     </keyIdentifier>
    </riID>
    <nonce>1ue3ue3ue10ue2109ue1</nonce>
   <roapURL>http://ri.example.com/ro.cgi?tid=qw683hgew7c</roapURL>
  </extendedTrigger>
</roap:roapTrigger>
```

H.1.20 Leave Domain Trigger

```
<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="1.1">
  <leaveDomain id="de32r23r4">
    <riID>
      <keyIdentifier xsi:type="roap:X509SPKIHash">
        <hash>aXENc+Um/9/NvmYKiHDLaErK0fk=</hash>
     </keyIdentifier>
    </riID>
    <nonce>sdfknjvfda438790fdjkl4rq</nonce>
    <roapURL>http://ri.example.com/ro.cgi?tid=qw683hgew7d</roapURL>
    <domainID>Domain-XYZ-001</domainID>
  </leaveDomain>
  <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>
```

H.1.21 Extended Leave Domain Trigger

```
<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="1.1">
  <extendedTrigger id="de32r23r5" type="leaveDomain">
    <riID>
      <keyIdentifier xsi:type="roap:X509SPKIHash">
        <hash>aXENc+Um/9/NvmYKiHDLaErK0fk=</hash>
     </keyIdentifier>
    </riID>
    <nonce>eqijdwqijdwq09uwqwqi</nonce>
    <roapURL>http://ri.example.com/ro.cgi?tid=qw683hgew7e</roapURL>
    <trgLeaveDomain>
     <deviceID>
        <keyIdentifier xsi:type="roap:X509SPKIHash">
            <hash>vXENc+Um/9/NvmYKiHDLaErK0gk=</hash>
        </keyIdentifier>
     </deviceID>
     <domainID>Domain-XYZ-002</domainID>
   </trgLeaveDomain>
  </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="#de32r23r5">
```

```
<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>id8JtuNKs12=drf5slo5hb+OM+3df=</ds:DigestValue>
   </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>Um/9/NvmYKiHDLaErK0fkj6lwx3r</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-002</roap:domainID>
  </ds:KeyInfo>
  <xenc:CipherData>
   <xenc:CipherValue>ufds32fdsorew9ufdsoi09krew90uderty5346wqurew</xenc:CipherValue>
  </xenc:CipherData>
 </encKey>
</roap:roapTrigger>
```

H.2 HTTP Transport Mapping Examples

H.2.1 Separate Delivery of DCF and Rights Object

This first example is a basic use case assuming only minimal integration between RI and CI (exchange of CEK and content ID prior to content and Rights Object delivery).

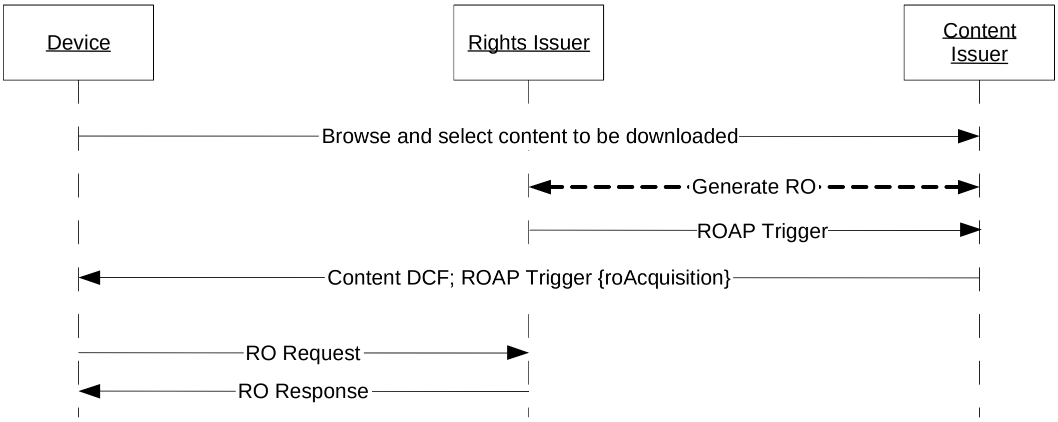
Device
Rights Issuer
Content Issuer
Browse and select content to be downloaded
Generate RO
ROAP Trigger
Content DCF; ROAP Trigger {roAcquisition}
RO Request
RO Response

Figure 28: Separate Delivery of DCF and RO

- A user browses through a content portal, selects content and so on.
- A Rights Object is generated for the purchase transaction using some backend interaction between RI and CI.
- RI generates a ROAP Trigger to initiate acquisition of the Right Object and delivers the ROAP Trigger to CI.
- The CI returns an HTTP Response containing a multipart/related. One entity is the content DCF, the other entity is the ROAP Trigger.

```
HTTP 1.1 200 OK
 Server: CoolServer/1.3.12
 Content-Type: multipart/related; boundary=”XX---XX”;
type=”application/vnd.oma.drm.dcf”

--XX---XX
 Content-Length: 1232
 Content-Type: application/vnd.oma.drm.dcf
... [DCF] ...
--XX---XX
 Content-Length: 986
 Content-Type: application/vnd.oma.drm.roap-trigger+xml
... [ROAP Trigger XML document] ...
--XX---XX--
```

- The ROAP Trigger is used by the DRM Agent on the Device to initiate a ROAP session to download a Rights Object. The DRM Agent issues an HTTP POST to the URL specified by the ROAP Trigger. The POST includes a ROAP-RORequest PDU in the HTTP request body.

```
POST http://www.acme.com/ro.cgi?roID=qw683hgew7d
 Host: www.acme.com
 User-Agent: CoolPhone/1.4
 Accept: application/vnd.oma.drm.roap-pdu+xml, multipart/related
 Accept-Charset: utf-8
 Content-Length: 125
 Content-Type: application/vnd.oma.drm.roap-pdu+xml

... [ROAP PDU] ...
```

An established RI Context is assumed in the example. If this were not the case, then the ROAP-RORequest would be preceded by a ROAP Registration transaction.

- The Rights Issuer returns an HTTP response containing a ROAP-ROResponse PDU in the HTTP response body.

```
HTTP 1.1 200 OK
 Server: CoolServer/1.3.12
 Content-Length: 986
 Content-Type: application/vnd.oma.drm.roap-pdu+xml

... [ROAP PDU] ...
```

H.2.2 Combined Delivery of DCF and Rights Object

This second example is a variation on the previous example with a closer relationship with RI and CI.

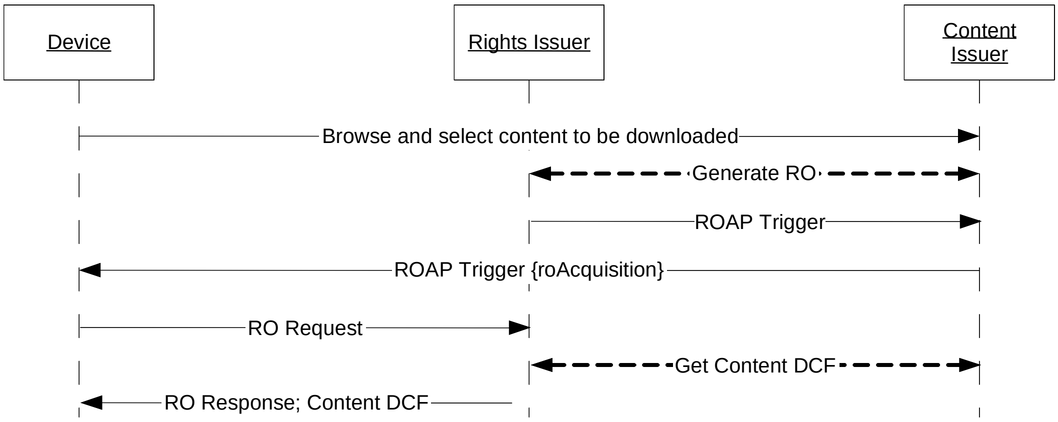

Figure 29: Combined Delivery of DCF and RO

1. A user browses through a content portal, selects content and so on.

2. A Rights Object is generated for the purchase transaction using some backend interaction between RI and CI.

3. RI generates a ROAP Trigger to initiate acquisition of the Right Object and delivers the ROAP Trigger to CI.

4. The CI returns an HTTP Response containing the ROAP Trigger.

```
HTTP 1.1 200 OK
 Server: CoolServer/1.3.12
 Content-Length: 986
 Content-Type: application/vnd.oma.drm.roap-trigger+xml

... [ROAP Trigger] ...
```

5. The ROAP Trigger is used by the DRM Agent on the Device to initiate a ROAP session to download a Rights Object. The POST includes a ROAP-RORequest PDU in the HTTP request body.

```
POST http://www.acme.com/ro.cgi?roID=qw683hgew7d
 Host: www.acme.com
 User-Agent: CoolPhone/1.4
 Accept: application/vnd.oma.drm.roap-pdu+xml, multipart/related
 Accept-Charset: utf-8
 Content-Length: 125
 Content-Type: application/vnd.oma.drm.roap-pdu+xml

... [ROAP PDU] ...
```

A previously established RI Context is assumed in the example. If this were not the case, then the ROAP-RORequest would be preceded by a ROAP-Registration transaction.

6. The Rights Issuer interacts with the CI to retrieve the DCF, and returns a multipart HTTP response containing as one entity a ROAP-ROResponse PDU, and as another entity the content object (DCF).

```
HTTP 1.1 200 OK
 Server: CoolServer/1.3.12
 Content-Type: multipart/related; boundary=”XX---XX”;
type=”application/vnd.oma.drm.roap-pdu+xml”

--XX---XX
 Content-Length: 986
 Content-Type: application/vnd.oma.drm.roap-pdu+xml
... [ROAP PDU] ...
--XX---XX
 Content-Length: 1232
 Content-Type: application/vnd.oma.drm.dcf
... [DCF] ...
--XX---XX–
```

H.2.3 Silent RO Acquisition Triggered by DCF Headers

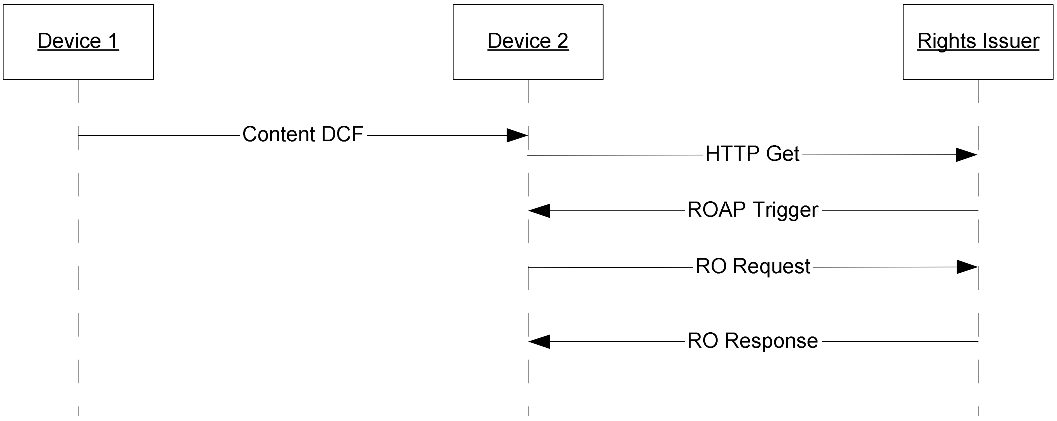

Figure 30: Silent RO Acquisition Triggered by DCF Headers

In this case a DCF is superdistributed to a Device, and the DRM Agent uses DCF headers to initiate a ROAP transaction and download a Rights Object.

– A user receives a DCF from another Device, e.g. through MMS, peer-to-peer, removable media, or some other transfer mechanism.

– If the DCF contains either a Silent header or a Preview header, then the DRM Agent attempts to request a Rights Object automatically. If the DRM Agent has an existing RI Context for the Rights Issuer, and has obtained user consent to request Rights Objects from the Rights Issuer, then the DRM Agent may proceed silently without further user interaction.

```
The DRM Agent sends an HTTP Get to the URL specified by the Silent or
Preview headers.  GET /ro.cgi?cid=qw683hgew7d HTTP/1.1
Host: www.acme.com
```

– The Rights Issuer returns an HTTP response containing a ROAP Trigger.

```
HTTP 1.1 200 OK
 Server: CoolServer/1.3.12
 Content-Length: 986
 Content-Type: application/vnd.oma.drm.roap-trigger+xml

... [ROAP Trigger] ...
```

– The ROAP Trigger is used by the DRM Agent on the Device to initiate a ROAP session to download a Rights Object. The POST includes a ROAP-RORequest PDU in the HTTP request body.

```
POST http://www.acme.com/ro.cgi?roID=qw683hgew7d
 Host: www.acme.com
 User-Agent: CoolPhone/1.4
 Accept: application/vnd.oma.drm.roap-pdu+xml, multipart/related
 Accept-Charset: utf-8
 Content-Length: 125
 Content-Type: application/vnd.oma.drm.roap-pdu+xml

... [ROAP PDU] ...
```

A previously established RI Context is assumed in the example. If this were not the case, then the ROAP-RORequest would be preceded by a ROAP-Registration transaction.

– The Rights Issuer returns an HTTP response containing a ROAP-ROResponse PDU in the HTTP response body.

```
HTTP 1.1 200 OK
 Server: CoolServer/1.3.12
 Content-Length: 986
 Content-Type: application/vnd.oma.drm.roap-pdu+xml

... [ROAP PDU] ...
```

H.3 Download OTA Examples

H.3.1 Separate Delivery of DRM Content and Rights Object

A Service Provider may use OMA Download OTA to deliver both the DRM Content and the Rights Object in separate transactions. The following figure shows the interaction between the logical system components residing in the Device and logical server components hosted by the Service Provider during the separate delivery of DRM Content and Rights Objects. Note that in the download transaction for retrieving the Rights Objects, the ROAP Trigger is co-delivered with the Download Descriptor using OMA Download OTA co-delivery method.

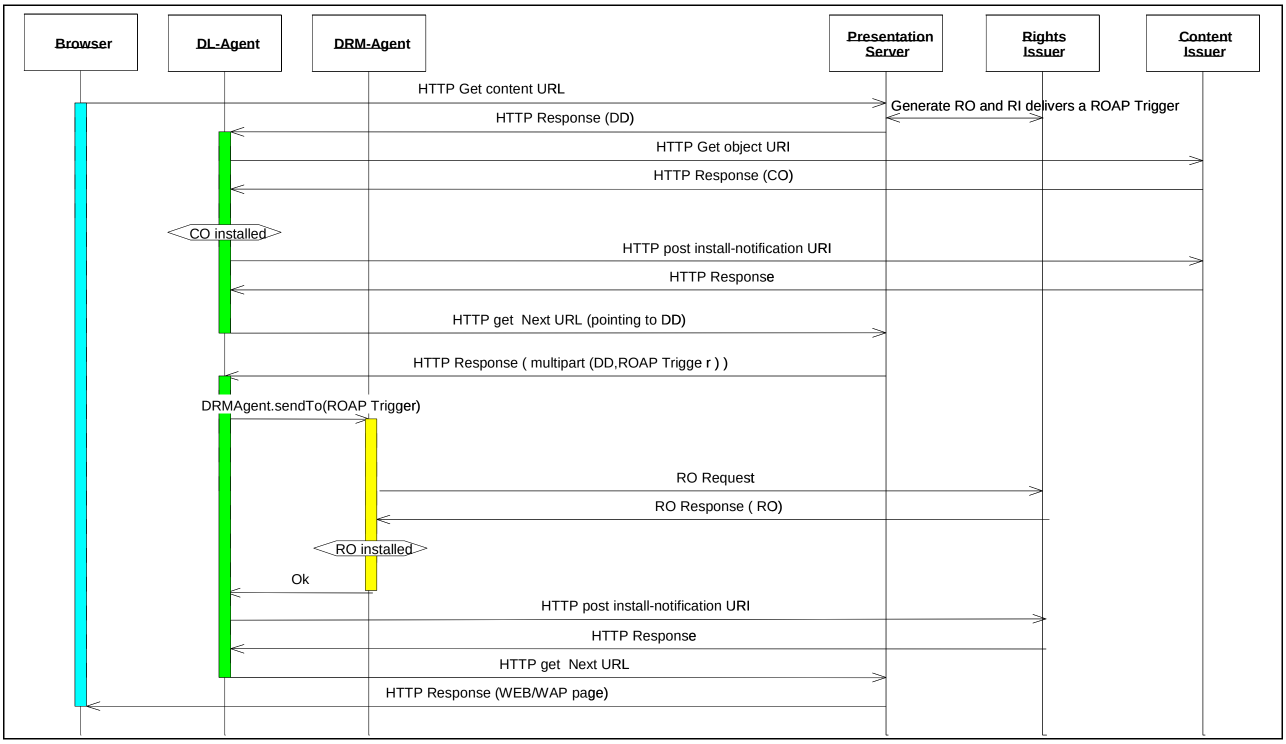
Browser
DL-Agent
DRM-Agent
Presentation Server
Rights Issuer
Content Issuer
HTTP Get content URL
Generate RO and RI delivers a ROAP Trigger
HTTP Response (DD)
HTTP Get object URI
HTTP Response (CO)
CO installed
HTTP post install-notification URI
HTTP Response
HTTP get Next URL (pointing to DD)
HTTP Response (multipart (DD,ROAP Trigge r))
DRMAgent.sendTo(ROAP Trigger)
RO Request
RO Response (RO)
RO installed
Ok
HTTP post install-notification URI
HTTP Response
HTTP get Next URL
HTTP Response (WEB/WAP page)

Figure 31: Using Download OTA to deliver DRM Content and Rights Object

1. A user browses through a content portal, selects content and so on. A Rights Object is generated for the purchase transaction using some backend interaction between RI and Presentation Server. RI generates a ROAP Trigger to initiate acquisition of the Right Object and delivers the ROAP Trigger to Presentation Server. When it is time to deliver content, the server returns an HTTP Response with a Download Descriptor (DD). The DD might, for example, point to a DCF file containing a JPEG image.

```
HTTP 1.1 200 OK
 Server: CoolServer/1.3.12
 Content-Length: 1232
 Content-Type: application/vnd.oma.dd+xml; charset=utf-8

<media xmlns="http://www.openmobilealliance.org/xmlns/dd">
  <type>application/vnd.oma.drm.dcf</type>
  <type>image/jpeg</type>
  <objectURI>http:/download.example.com/image.dcf</objectURI>
  <size>100</size>
  <installNotifyURI>
    http://download.example.com/notify?tid=2h3jh3g4
  </installNotifyURI>
  <nextURL>
    http://ri.example.com/ro?tid=2h3jh3g4
  </nextURL>
</media>
```

2. The Download Agent requests the Content using the *objectURI*.

```
GET /image.dcf HTTP/1.1
Host: download.example.com
Accept: image/gif, image/jpg, application/vnd.oma.drm.dcf
```

3. The DCF is returned to the Download Agent.

```
HTTP/1.1 200 OK
Server: CoolServer/1.3.12
Content-Length: 1234
Content-Type: application/vnd.oma.drm.dcf

… DCF containing JPEG picture…
```

4. The Download Agent installs the Content and posts an installation notification.

```
POST /notify?tid=2h3jh3g4 HTTP/1.1
 Host: download.example.com
 Content-Length: 13
900 Success
```

5. In this example the DD for the DCF specifies a *nextURL*. This means that when the Download Agent is done downloading and installing the DCF, it will automatically issue an HTTP GET to the URL specified by the *nextURL* DD parameter. This can be used to seamlessly redirect the Device from the CI to the RI.

```
GET /ro?tid=2h3jh3g4 HTTP/1.1
Host: ri.example.com
```

6. The RI returns a DD and the ROAP Trigger.

```
HTTP 1.1 200 OK
 Server: CoolServer/1.3.12
 Content-Type: multipart/related; boundary=”XX---XX”;
type=”application/vnd.oma.dd+xml”
 --XX---XX
 Content-Length: 1232
 Content-Type: application/vnd.oma.dd+xml; charset=utf-8

<media xmlns="http://www.openmobilealliance.org/xmlns/dd">
  <type>application/vnd.oma.drm.roap-pdu+xml</type>
  <type>application/vnd.oma.drm.ro+xml</type>
  <objectURI>cid:w087w78087sdf80@ri.example.com</objectURI>
  <size>1232</size>
  <installNotifyURI>
    http://ri.example.com/notify?tid=2h3jh3g4
  </installNotifyURI>
  <nextURL>
    http://provider.example.com/trans_complete.html
  </nextURL>
</media>
```

```
--XX---XX
Content-Length: 986
Content-ID: <w087w78087sdf80@ri.example.com>
Content-Type: application/vnd.oma.drm.roap-trigger+xml
<roapTrigger>
  <roAcquisition>
      <riID>
       <keyIdentifier xsi:type="roap:X509SPKIHash">
         <hash>aXENc+Um/9/NvmYKiHDLaErK0fk=</hash>
       </keyIdentifier>
     </riID>
      <roapURL>http://ri.example.com/ro.cgi?tid=qw683hgew7d</roapURL>
     <roID>239087dsf78</roID>
  </roAcquisition>
</roapTrigger>
--XX---XX
```

7. The ROAP Trigger is used by the DRM Agent on the Device to initiate a ROAP session to download a Rights Object. The DRM Agent issues an HTTP POST to the ROAP Trigger URL. The POST includes a ROAP-RORequest PDU in the HTTP request body.

```
POST /ro.cgi?tid=qw683hgew7d HTTP/1.1
 Host: ri.example.com
 User-Agent: CoolPhone/1.4
 Accept: application/vnd.oma.drm.roap-pdu+xml,multipart/related
 Content-Length: 125
 Content-Type: application/vnd.oma.drm.roap-pdu+xml

... [ROAP PDU] ...
```

8. The RI returns the ROAP-ROResponse PDU.

```
HTTP 1.1 200 OK
 Server: CoolServer/1.3.12
 Content-Length: 986
 Content-Type: application/vnd.oma.drm.roap-pdu+xml

... [ROAP PDU] ...
```

9. The DRM Agent processes the ROAP PDU and sends the installation status (success or failure) to the Download Agent. The Download Agent sends the installation status to the RI using the *installNotifyURI*.

```
POST /notify?tid=2h3jh3g4 HTTP/1.1
 Host: ri.example.com
 Content-Length: 13
900 Success
```

10. The Download Agent immediately navigates to the *nextURL*.

```
GET /trans_complete.html HTTP/1.1
Host: provider.example.com
```

H.3.2 Combined Delivery of Content DCF and Rights Object

This example is an extension to the previous example, assuming a closer relationship between the RI and CI allowing the content DCF and the RO to be delivered together in a single OMA Download OTA transaction. Also in this example, the ROAP Trigger is co-delivered with the Download Descriptor using the OMA Download OTA co-delivery method.

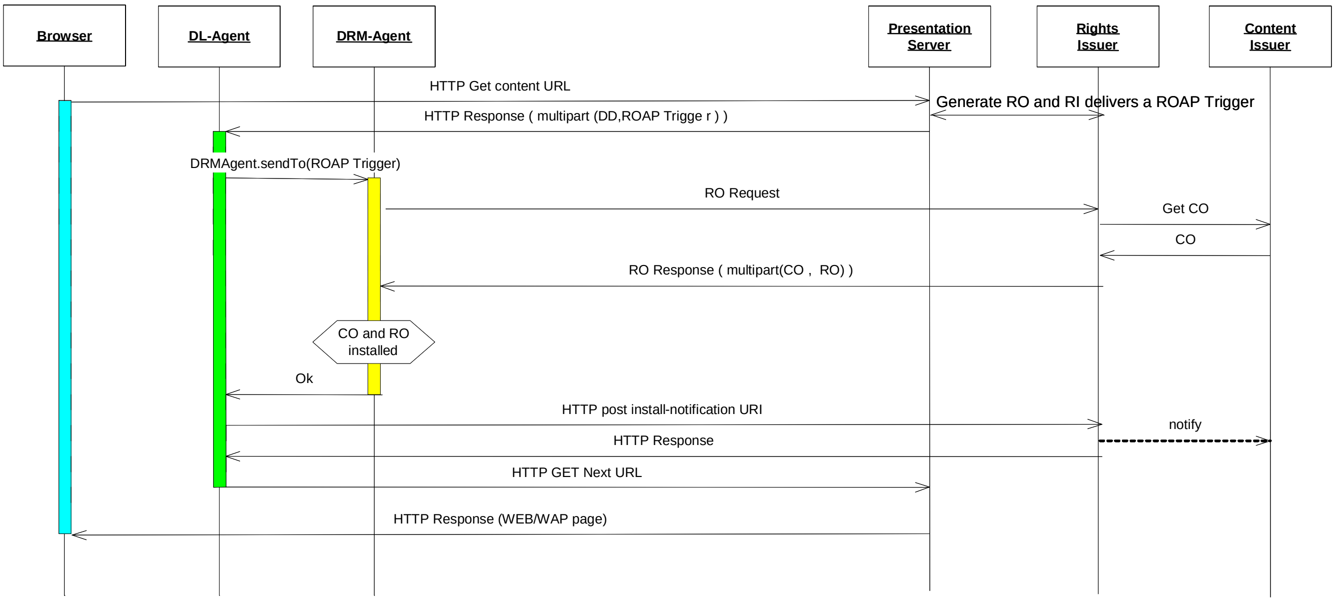
Browser
DL-Agent
DRM-Agent
Presentation Server
Rights Issuer
Content Issuer
HTTP Get content URL
Generate RO and RI delivers a ROAP Trigger
HTTP Response (multipart (DD,ROAP Trigge r))
DRMAgent.sendTo(ROAP Trigger)
RO Request
Get CO
CO
RO Response (multipart(CO , RO))
CO and RO installed
Ok
HTTP post install-notification URI
notify
HTTP Response
HTTP GET Next URL
HTTP Response (WEB/WAP page)

Figure 32: Combined Delivery of DRM Content and Rights Object

1. A user browses through a content portal, selects content and so on. A Rights Object is generated for the purchase transaction using some backend interaction between RI and Presentation Server. RI generates a ROAP Trigger to initiate acquisition of the Right Object and delivers the ROAP Trigger to Presentation Server. When it is time to deliver content, the server returns a DD and the ROAP Trigger to initiate delivery of the combined Rights Object and DRM Content.

```
HTTP 1.1 200 OK
 Server: CoolServer/1.3.12
Content-Type: multipart/related; boundary=”XX---XX”;
type=”application/vnd.oma.dd+xml”
 --XX---XX
 Content-Length: 1232
 Content-Type: application/vnd.oma.dd+xml; charset=utf-8

<media xmlns="http://www.openmobilealliance.org/xmlns/dd">
  <type>application/vnd.oma.drm.roap-pdu+xml</type>
  <type>application/vnd.oma.drm.ro+xml</type>
  <type>application/vnd.oma.drm.dcf</type>
  <objectURI>cid:sd8963234l@ri.example.com</objectURI>
  <size>2118</size>
  <installNotifyURI>
    http://ri.example.com/notify?tid=2h3jh3g4
  </installNotifyURI>
  <nextURL>
    http://provider.example.com/trans_complete.html
  </nextURL>
</media>
--XX---XX
Content-Length: 986
Content-ID: <sd8963234l@ri.example.com>
Content-Type: application/vnd.oma.drm.roap-trigger+xml
<roapTrigger>
   <roAcquisition>
      <riID>
       <keyIdentifier xsi:type="roap:X509SPKIHash">
         <hash>aXENc+Um/9/NvmYKiHDLaErK0fk=</hash>
       </keyIdentifier>
      </riID>
      <roID>239087dsf78</roID>
      <roapURL>http://ri.example.com/ro.cgi?tid=g97sd976s90</roapURL>
   </roAcquisition>
</roapTrigger>
--XX---XX
```

2. The ROAP Trigger is used by the DRM Agent on the Device to initiate a ROAP session to download the combined Rights Object and DRM Content. The DRM Agent issues an HTTP POST to the URL specified by the ROAP Trigger. The POST includes a ROAP-RORequest PDU in the HTTP request body.

```
POST /ro.cgi?tid=g97sd976s90 HTTP/1.1
 Host: ri.example.com
 User-Agent: CoolPhone/1.4
 Accept: application/vnd.oma.drm.roap-pdu+xml, multipart/related
 Content-Length: 125
 Content-Type: application/vnd.oma.drm.roap-pdu+xml

... [ROAP PDU] ...
```

3. The RI returns a multipart containing a ROAP-ROResponse PDU and the DRM Content.

```
HTTP 1.1 200 OK
 Server: CoolServer/1.3.12
 Content-Type: multipart/related; boundary=”XX---XX”;
type=”application/vnd.oma.drm.roap-pdu+xml”

--XX---XX
 Content-Length: 986
 Content-Type: application/vnd.oma.drm.roap-pdu+xml
... [ROAP PDU] ...
--XX---XX
 Content-Length: 1232
 Content-Type: application/vnd.oma.drm.dcf
... [DCF] ...
--XX---XX--
```

4. The DRM Agent installs the Rights Object and DRM Content. The DRM Agent notifies the Download Agent of installation success. The Download Agent posts the installation notification.

```
POST /notify?tid=2h3jh3g4 HTTP/1.1
 Host: ri.example.com
 Content-Length: 13

900 Success
```

H.3.3 Device Identification

As part of a browsing session a Service Provider may use OMA Download OTA to deliver an Identification ROAP Trigger. The following figure shows the interaction between the logical system components residing in the Device and logical server components hosted by the Service Provider during delivery of an Identification Trigger.

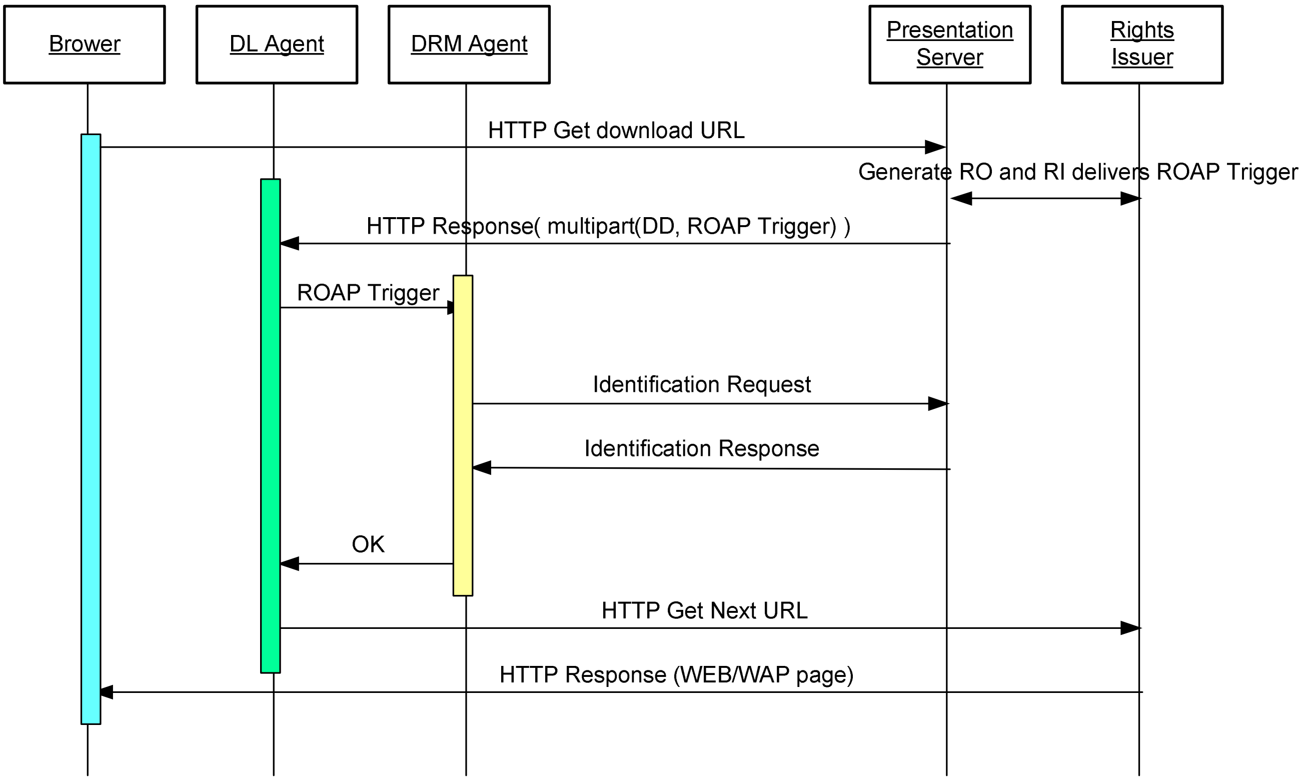

Figure 33: Using Download OTA to deliver an Identification ROAP Trigger

1. A user browses through a content portal, selects the download section. The RI generates an Identification ROAP Trigger and delivers the ROAP Trigger to Presentation Server.

2. The RI returns a DD and the ROAP Trigger.

```
HTTP 1.1 200 OK
 Server: CoolServer/1.3.12
 Content-Type: multipart/related; boundary=”XX---XX”;
type=”application/vnd.oma.dd+xml”
 --XX---XX
 Content-Length: 1232
 Content-Type: application/vnd.oma.dd+xml; charset=utf-8

<media xmlns="http://www.openmobilealliance.org/xmlns/dd">
  <type>application/vnd.oma.drm.roap-pdu+xml</type>
  <type>application/vnd.oma.drm.ro+xml</type>
  <objectURI>cid:w087w78087sdf80@ri.example.com</objectURI>
  <size>1232</size>
  <nextURL>
    http://provider.example.com/trans_complete.html
  </nextURL>
</media>
--XX---XX
Content-Length: 986
Content-ID: <w087w78087sdf80@ri.example.com>
Content-Type: application/vnd.oma.drm.roap-trigger+xml
<roapTrigger>
  <identificationRequest>
      <riID>
       <keyIdentifier xsi:type="roap:X509SPKIHash">
```

```
         <hash>aXENc+Um/9/NvmYKiHDLaErK0fk=</hash>
       </keyIdentifier>
     </riID>
      <roapURL>http://ri.example.com/id.cgi?tid=qw683hgew7d</roapURL>
  </identificationRequest>
</roapTrigger>
--XX---XX
```

3. The ROAP Trigger is used by the DRM Agent on the Device to initiate a 2-pass Identification ROAP session. The DRM Agent issues an HTTP POST to the ROAP Trigger URL. The POST includes a ROAP- IdentificationRequest PDU in the HTTP request body.

```
POST /id.cgi?tid=qw683hgew7d HTTP/1.1
 Host: ri.example.com
 User-Agent: CoolPhone/1.4
 Accept: application/vnd.oma.drm.roap-pdu+xml,multipart/related
 Content-Length: 125
 Content-Type: application/vnd.oma.drm.roap-pdu+xml

... [ROAP PDU] ...
```

4. The RI returns the ROAP- IdentificationResponse PDU.

```
HTTP 1.1 200 OK
 Server: CoolServer/1.3.12
 Content-Length: 986
 Content-Type: application/vnd.oma.drm.roap-pdu+xml

... [ROAP PDU] ...
```

5. The Download Agent immediately navigates to the *nextURL*.

```
GET /trans_complete.html HTTP/1.1
Host: provider.example.com
```

H.4 MMS Examples

H.4.1 MMS delivery of DCF within a SMIL presentation

The example MMS message shown below contains a presentation description in the form of a SMIL document. From within this document the second body part of the message is referenced by a Content-ID, which is associated with the referenced part in the multipart structure in the form of a header field (containing the ID “same-reference-as-in-SMIL-doc” in this example). This is explicitly different from the Content-ID included in the DCF (“same-reference-as-used-in-associated-ROs”) which serves as a reference for the Rights Object(s) associated with the Media Object. As an alternative to the Content-ID in the multipart structure a Content-Location may be used according to [MMSENC]. For a better understanding, the example below is illustrated in textual format, although the MMS PDUs are binary encoded on the interface between the MMS Proxy-Relay and the MMS Client according to [MMSENC].

```
From:customer@mmsprovider.com
To:anothercustomer@anothermmsprovider.com
Subject:MMS message with DRM content
X-MMS-Version:1.2
...[More MMS headers]
Content-Type:multipart/related;boundary=firststring;start=secondstring;
type=”application/smil”

--firststring
Content-ID:secondstring
Content-Type:application/smil
...[SMIL doc]
```

```
--firststring
Content-ID:same-reference-as-in-SMIL-doc
Content-Type: application/vnd.oma.drm.dcf
...[DCF containing Content-ID:same-reference-as-used-in-associated-ROs]
--firststring--
```

H.5 ROAP over OBEX Examples

Example messages between the Connected Device and the Unconnected Device are illustrated in this section utilizing ROAP over OBEX transport mapping.

H.5.1 ROAP Trigger

This message is sent from the Connected Device to the Unconnected Device after:

The Connected Device has received the trigger from the RI;

The Connected Device has determined that the ROAP Trigger is not for itself; and

The Connected Device has established a directed OBEX connection to the Unconnected Device's OBEX server.

Bytes	Meaning
0x82	Opcode PUT, single packet request, final bit set
0x0301	Packet length (a total of 769 bytes in this case)
0xCB	Connection Id HI
0x00000001	ConnectionId = 1
0x42	Type HI
0x0027	Total length of Type header (including HI and length fields)
"application/vnd.oma.roap-trigger+xml"	Type of object, null terminated ASCII text
0x49	End-of-Body HI
0x02D2	Length of body (trigger) is 719 bytes (= whole object)(+ 3 bytes header information)
0x….	The ROAP-JoinDomain trigger goes here…

H.5.2 ROAP-OBEX Server Response

This is the response message from the Unconnected Device, sent by that Device's OBEX server.

Bytes	Meaning
0xA0	Opcode SUCCESS, Final bit set
0x016B	Length of response packet (363 bytes)
0xCB	Connection Id HI
0x00000001	ConnectionId = 1
0x42	Type HI
0x001F	Total length of Type header (28 bytes + 3 bytes header information)
"application/vnd.oma.roap-pdu+xml"	Type of object, null terminated ASCII
0x49	End-of-Body HI
0x0144	Body header length (321 bytes + 3 bytes header information)
0x….	The triggered ROAP request goes here

H.6 Example – Exporting to Removable Media

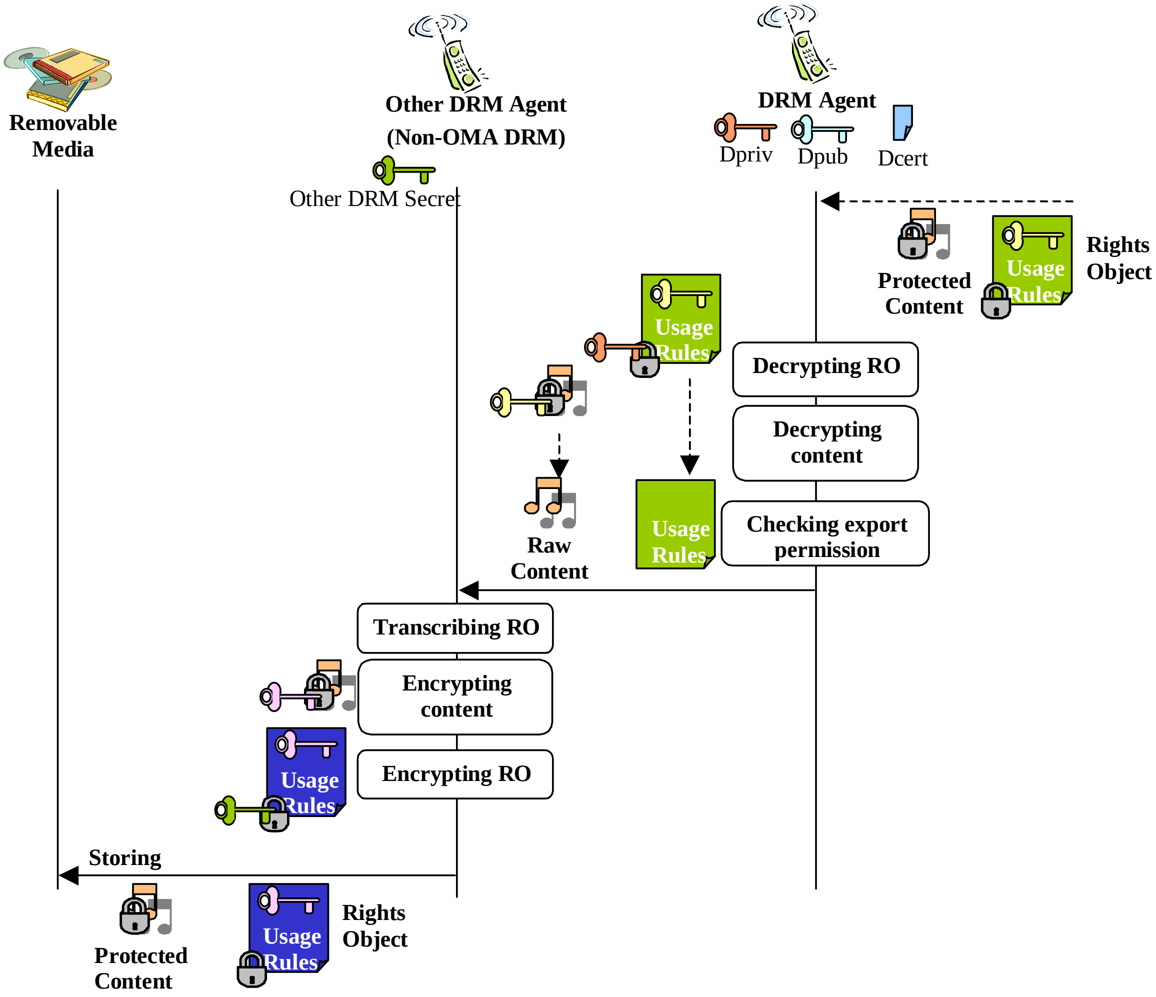

Figure 34: Example – Exporting to Removable Media

An example of exporting DRM Content and Rights Object to other DRM (non-OMA DRM) system, which has some authorised protection mechanism, is shown above.

After DRM Content and protected Rights Object are delivered to a trusted OMA DRM Agent, the DRM Content is consumed by the OMA DRM Agent according to permissions and constraints described in the protected Rights Object. When consuming the content, OMA DRM Agent decrypts the protected RO with DRM Agent private key and decrypts the DRM Content with CEK that is derived from decrypted Rights Object.

When exporting, OMA DRM Agent checks permissions described in the Rights Object whether Rights Issuer allows the content to be exported to targeted DRM system, whether its content type is appropriate and whether its usage rules are compatible with targeted DRM system.

When user wants to download exportable content and Rights Issuer notices it in the course of content discovery interaction, it would be expected that both of the DRM Content and RO are suitable for the targeted DRM capability.

The raw content and usage rules are transferred from OMA DRM Agent to the other DRM Agent.

The other DRM Agent transcribes the compatible usage rules to the other DRM usage rules according to the general rule and the specific rule defined by the other DRM system and Rights Issuer to maintain consistency of the Rights Object.

Sample transcription rules are:
 Any other permissions MUST NOT be granted.
 Any existing constraints MUST NOT be ignored.
 Default permissions and constraints MAY be supplied.

Even stateful Rights Object could be transcribed and exported to other DRM system if those rules allow it.

The other DRM Agent creates new CEK inside, and encrypts the content with the new CEK and encrypts transcribed usage rules including the CEK with other DRM system's secret key.

The other DRM Agent and the removable media authenticate with each other to make sure that they are trusted, and stores the encrypted content and the usage rules onto a removable media according to the other DRM specific format.

Then user can pull out the removable media from the Device, insert it to other DRM compliant Device such as portable music player, to enjoy playing the content.

The two DRM Agents, OMA DRM Agent and the other DRM Agent, may reside in a single Device or different Devices, but these two agents and data channel between the two have to be implemented in a secure manner according to some compliance rules or robustness rules which may be defined related to a specific service, by Rights Issuers, Service Providers, and Device Manufacturers who participate in the service.

H.7 WBXML Encoding

H.7.1 ROAP Trigger

This example shows the annotated WBXML encoding of a ROAP Trigger. Note that the trigger is in Exclusive Canonical form. To improve the readability, whitespace has been added between the elements, but it should be understood that the trigger that is encoded into WBXML has no whitespace between the elements (should whitespace be present, then that would be encoded as opaque data).

```
<roap:roapTrigger xmlns:roap="urn:oma:bac:dldrm:roap-1.0" version="1.1">
  <leaveDomain id="de32r23r4">
    <riID>
      <keyIdentifier xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="roap:X509SPKIHash">
        <hash>IMUjt0T7l2mSweqebvQyZp89fqI=</hash>
      </keyIdentifier>
    </riID>
    <nonce>SnVzdCBBIE5vbmNlICE=</nonce>
    <roapURL>http://ri.example.com/ro.cgi?tid=qw683hgew7d</roapURL>
    <domainID>Domain-XYZ-001</domainID>
  </leaveDomain>
  <signature>
    <ds:SignedInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"></ds:CanonicalizationMethod>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-
sha1"></ds:SignatureMethod>
      <ds:Reference URI="#de32r23r4">
        <ds:Transforms>
          <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"></ds:Transform>
        </ds:Transforms>
        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></ds:DigestMethod>
        <ds:DigestValue>e9INNPV9OzBDdAkkvnSYLVUqg5o=</ds:DigestValue>
      </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">Tm8gSE1BQy1TSEExIHJlc3VsdCE=</ds:SignatureValue
>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:RetrievalMethod URI="#K_MAC"></ds:RetrievalMethod>
    </ds:KeyInfo>
  </signature>
  <encKey Id="K_MAC">
    <xenc:EncryptionMethod xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"></xenc:EncryptionMethod>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <roap:domainID>Domain-XYZ-001</roap:domainID>
    </ds:KeyInfo>
    <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
      <xenc:CipherValue>Tm90IGEgd3JhcHBlZCBNQUMga2V5ISEh</xenc:CipherValue>
    </xenc:CipherData>
  </encKey>
</roap:roapTrigger>
```

This is encoded to the following WBXML document:

WBXML code (hexadecimal)	Description
03	WBXML version number – WBXML version 1.3
13	Public identifier (-//OMA// DRM 2.1//EN)
6A	Charset = UTF-8
00	String table length = 00 (empty string table)
C5	<roap:roapTrigger
06	xmlns:roap=
85	"urn:oma:bac:dldrm:roap-1.0" (attribute value)
0D	version=
90	"1.1" (attribute value)
01	>
C9	<leaveDomain
0F	id=
03	STR_I (inline string followed by a terminator)
64 65 33 32 72 32 33 72 34 00	de32r23r4 + string terminator
01	>
4C	<riID>
D7	<keyIdentifier
07	xmlns:xsi=
8B	"http://www.w3.org/2001/XMLSchema-instance"
05	xsi:type=
8C	"roap:X509SPKIHash"
01	>
58	<hash> The <hash>…</hash> value is originally base64 encoded which is simply encoded as an inline string.
03	STR_I (inline string followed by a terminator)
49 4d 55 6a 74 30 54 37 6c 32 6d 53 77 65 71 65 62 76 51 79 5a 70 38 39 66 71 49 3d 00	Base64binary hash value: ‘IMUjt0T7l2mSweqebvQyZp89fqI=’
01	</hash>
01	</keyIdentifier>

01	</riID>
4E	<nonce>
03	STR_I (inline string followed by a terminator)
53 6e 56 7a 64 43 42 42 49 45 35 76 62 6d 4e 6c 49 43 45 3d 00	Base64binary \nonce value: 'SnVzdCBBIE5vbmNlICE='
01	</nonce>
4F	<roapURL>
03	STR_I string literal, followed by a terminator
68 74 74 70 3a 2f 2f 72 69 2e 65 78 61 6d 70 6c 65 2e 63 6f 6d 2f 72 6f 2e 63 67 69 3f 74 69 64 3d 71 77 36 38 33 68 67 65 77 37 64 00	http://ri.example.com/ro.cgi?tid=qw683hgew7d
01	</roapURL>
50	<domainID>
03	STR_I string literal, followed by a terminator
44 6f 6d 61 69 6e 2d 58 59 5a 2d 30 30 31 00	Domain-XYZ-001
01	</domainID>
01	</leaveDomain>
4A	<signature>
D9	<ds:SignedInfo
09	xmlns:ds=
89	"http://www.w3.org/2000/09/xmldsig#"
01	>
90	<ds:CanonicalizationMethod
12	Algorithm=
8E	"http://www.w3.org/2001/10/xml-exc-c14n#"
01	></ds:CanonicalizationMethod> Note that because this tag was started as 'No content, with attributes', it means that the termination of the attribute list also signals the termination of the element, hence the presence of the end tag in the canonical XML document.
91	<ds:SignatueMethod
12	Algorithm=
8F	"http://www.w3.org/2000/09/xmldsig#hmac-sha1"
01	></ds:SignatureMethod>
D2	<ds:Reference
13	URI=
03	STR_I (inline string followed by a terminator)
23 64 65 33 32 72 32 33 72 34 00	#de32r23r4 + string terminator
01	>
60	<ds:Transforms>
A3	<ds:Transform
12	Algorithm=
8E	"http://www.w3.org/2001/10/xml-exc-c14n#"
01	></ds:Transform>
01	</ds:Transforms>
A1	<ds:DigestMethod
12	Algorithm=
8D	"http://www.w3.org/2000/09/xmldsig#sha1"

01	></ds:DigestMethod>
62	<ds:DigestValue>
03	STR_I (inline string followed by a terminator)
65 39 6c 4e 4e 50 56 39 4f 7a 42 44 64 41 6b 6b 76 6e 53 59 4c 56 55 71 67 35 6f 3d 00	Equivalent to the base64Binary ‘e9lNNPV9OzBDdAkkvnSYLVUqg5o=’
01	</ds:DigestValue>
01	</ds:Reference>
01	</ds:SignedInfo>
DA	<ds:SignatureValue
09	xmlns:ds=
89	"http://www.w3.org/2000/09/xmldsig#"
03	STR_I (inline string followed by a terminator)
54 6d 38 67 53 45 31 42 51 79 31 54 53 45 45 78 49 48 4a 6c 63 33 56 73 64 43 45 3d 00	base64Binary signatureValue: ‘Tm8gSE1BQy1TSEExIHJlc3VsdCE=’
01	</ds:SignatureValue>
DB	<ds:KeyInfo
09	xmlns:ds=
89	"http://www.w3.org/2000/09/xmldsig#"
01	>
9F	<ds:RetrievalMethod
13	URI=
B3	"#K_MAC"
01	></ds:RetrievalMethod>
01	</ds:KeyInfo>
01	</signature>
CB	<encKey
10	Id=
B5	"K_MAC"
01	>
A4	<xenc:EncryptionMethod
12	Algorithm=
92	"http://www.w3.org/2001/04/xmlenc#kw-aes128"
01	></xenc:EncryptionMethod>
DB	<ds:KeyInfo
09	xmlns:ds=
89	"http://www.w3.org/2000/09/xmldsig#"
01	>
52	<roap:domainID>
03	STR_I string literal, followed by a terminator
44 6f 6d 61 69 6e 2d 58 59 5a 2d 30 30 31 00	Domain-XYZ-001
01	</roap:domainID>
01	</ds:KeyInfo>
E5	<xenc:CipherData
08	xmlns:xenc=
8A	"http://www.w3.org/2001/04/xmlenc#"
01	>
66	<xenc:CipherValue>

03	STR_I string literal, followed by a terminator
54 6d 39 30 49 47 45 67 64 33 4a 68 63 48 42 6c 5a 43 42 4e 51 55 4d 67 61 32 56 35 49 53 45 68 00	base64Binary cipherValue: ‘Tm90IGEgd3JhcHBlZCBNQUMga2V5ISEh’
01	</xenc:CipherValue>
01	</xenc:CipherData>
01	</encKey>
01	</roap:roapTrigger>

This trigger in WBXML format requires 346 bytes. In Exclusive Canonical form, it requires 1656 bytes. This compact representation achieves a savings of around 79%.

H.8 State information Examples for Uploading RO :

The example state information shown below shows how to report the current state information of a stateful RO to the RI in the ROAP-RO Upload request message.

The original stateful RO issued by the RI is :

```
<o-ex:rights
 xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
 xmlns:o-dd="http://odrl.net/1.1/ODRL-DD"
 xmlns:oma-dd="http://www.openmobilealliance.com/oma-dd"
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
 o-ex:id="C.2">
 <o-ex:context>
  <o-dd:version>2.1</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>DCFHash</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="REKReference"/>
     </ds:KeyInfo>
     <xenc:CipherData>
      <xenc:CipherValue>EncryptedCEK</xenc:CipherValue>
     </xenc:CipherData>
    </xenc:EncryptedKey>
   </ds:KeyInfo>
  </o-ex:asset>
  <o-ex:permission>
```

```
   <o-ex:constraint o-ex:id="C-1">
    <o-dd:count>100</o-dd:count>
   </o-ex:constraint>
   <o-dd:play>
    <o-ex:constraint o-ex:id="C-2">
     <o-dd:count>10</o-dd:count>
     <o-dd:interval>P20DT00H00M00S</o-dd:interval>
    </o-ex:constraint>
   </o-dd:play>
   <o-dd:display>
   </o-dd:display>
  </o-ex:permission>
 </o-ex:agreement>
</o-ex:rights>
```

Assuming that the first time the user played the content is October 01st, 2005, 00:00:00 UTC and the user has played the content 2 times before uploading. The Device converts the interval constraint to a dateTime constraint and the the <stateInfo> in the ROAP-RO Upload request message should be:

```
<stateInfo o-ex:id="C-1">
 < o-dd:count>98</o-dd:count>
</stateInfo >
<stateInfo o-ex:id="C-2">
     <o-dd:count>8</o-dd:count>
     <o-dd:datetime>
      <o-dd:end>2005-10-21T00:00:00Z</o-dd:end>
     </o-dd:datetime>
</stateInfo>
```

Appendix I. Import (Informative)

Very similar to export from OMA DRM (chapter 17), the user may wish to render content on a OMA DRM Device after having downloaded that protected content using another device using a non-OMA DRM system. Import is an operation in which the non-OMA DRM protected content and corresponding rights are transferred from a non-OMA DRM system to the OMA DRM system. The non-OMA DRM system controls whether or not to allow the import into OMA DRM.

The basic concept of import into OMA DRM from a non-OMA DRM system is illustrated below in Figure 35. OMA does not specify the exact rules for mapping rights from non-OMA DRM systems to OMA Rights. It is the responsibility of appropriate bodies governing the use the non-OMA DRM system to define the necessary mechanisms for mapping into OMA DRM Rights.

In OMA DRM 2.0, the entities to manage RO’s and DCF’s and their distribution to other devices are the Rights Issuer and Content Issuer. Based on this specification, import is therefore possible by developing an import user equipment that implements an OMA DRM Rights Issuer and Content Issuer and provides a secure environment for the transfer of content and rights from the non-OMA system to OMA. This import user equipment must be approved by the appropriate trust authorities. After the content has been imported into OMA DRM, it can be transferred to other Devices using ROAP and possibly using the Domain concept.

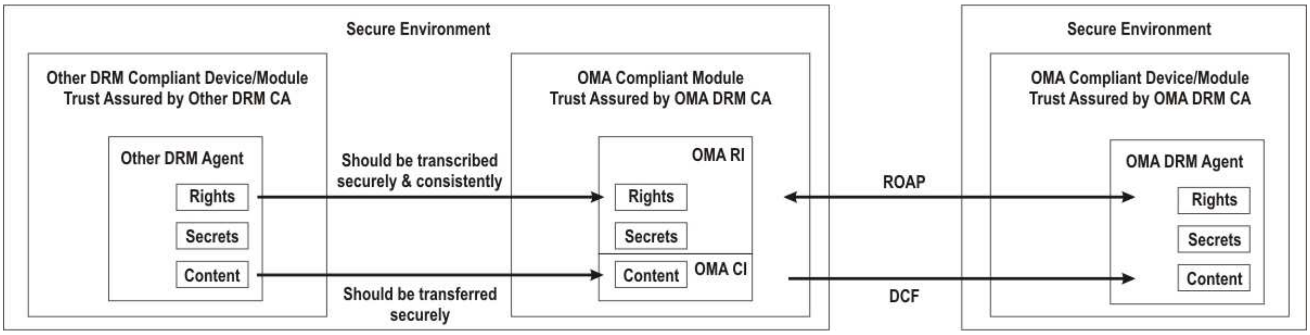

Figure 35 Example - Importing from a non-OMA DRM system

It must be stressed that the trust authorities must carefully consider the security threats involved with this deployment of functions. If the security of the environment in which the import takes place is compromised, then the OMA certificate of the import user equipment can be misused to create seemingly valid Rights Objects for pirated Content (“content laundering”), until the compromise is discovered and the certificate is revoked. Future versions of this enabler are expected to provide additional mechanisms to mitigate these threats.

This deployment of functions also requires the import user equipment to manage the domain in which the created Rights Objects are valid (if domains are used) and a proprietary way for the import user equipment to receive OCSP responses and other revocation information. Also these issues are expected to be addressed by future versions of this enabler.

Appendix J. 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 DRM Agent in case it encounters unknown extensions.

The XML schema of OMA DRM 2.1 explicitly enables forward compatibility using wildcards at selected locations. Future version of OMA DRM and also 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 OMA DRM 2.1 schema. Every conformant implementation of OMA DRM 2.1 will be able to parse the message or data structure and correctly deal with its content. The behaviour of an OMA DRM 2.1 DRM Agent that receives a message or data structure that does NOT validate against the OMA DRM 2.1 schema is undefined. It is possible that the message or data structure is discarded completely.

This appendix contains a number of examples of how the OMA DRM 2.1 wildcards can be used for future extensions that are correctly handled by OMA DRM 2.1 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 (extendedDomainInfo, roPayloadAliases, identificationRequest-trigger) may serve as examples.

J.1 ROPayload with future extensions

This example of a possible future ROPayload is based on example G.1.7. The future modifications are marked in a **bold** typeface. In this example, the ROPayload:

- is of version 2.3
- contains in addition to a play-permission (already define in OMA DRM 2.0) a “NewKindOfPermission”-permission (unknown in OMA DRM 2.1)
- some additional elements in the ROPayload, appended after all elements known from OMA DRM 2.0.

Implementations of OMA DRM 2.1 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>n8yu98hy0e2109eu09ewf09u</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 2.2/>
    <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> sIo5hb+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>
```

J.2 ROAP-PDU with future extensions

This example of a possible future joinDomainResponse is based on example G.1.9. The modifications are marked in a **bold** typeface. In this example, the ROAP-PDU contains two new extensions, unknown in OMA DRM 2.1. Implementations of OMA DRM 2.1 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>
```

J.3 ROAP Response with future status code

This example is of a possible future leaveDomainResponse is based on example G.1.11. The modifications are marked in a **bold** typeface. In this example, the leaveDomainResponse is unsuccesful, for a reason not specified by OMA DRM 2.1. 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>
```

J.4 New type of ROAP Trigger

This example is of a possible future ROAP Trigger is modified from example G.1.12. The modifications are marked in a **bold** typeface. Implementation of OMA DRM 2.1 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="1.1">
 <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> sIo5hb+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>
```

Appendix K. Media Transfer Protocol (Informative)

Media Transfer Protocol (MTP) enables the management and transfer of content on/to transient storage devices. MTP is an extension of the Picture Transfer Protocol (PTP), and is targeted primarily at digital still cameras, portable media players, and cellular phones. MTP provides support for many media and file formats, but it does not natively support the OMA DRM (P)DCF formats. This Appendix specifies how to add support in MTP compliant Devices for transfer/synchronization of (P)DCFs.

The MTP specification is published by the USB-IF [MTP]. This Appendix is intended to also apply to future versions of MTP.

K.1 MTP Object Format Codes

The overall intent is that content which is OMA DRM 2.x protected must be transferred in its (P)DCF container file, but is identified in MTP using the Object Format Code for the *ContentType* as specified in the (P)DCF *Discrete Media Headers* box (see [DRMCF-v2].

Although the content is identified by it's native content type an OMA DRM Vendor Extension (defined below) enables to "tag" the content as being OMA DRM V2 protected.

K.2 DRM Status Object Property

MTP defines an Object Property, DRM Status, with PropertyCode 0xDC9D. The property identifies the DRM status of the object (i.e. whether the object is protected or not). For OMA DRM 2.x protected content the DRM Status object property SHOULD indicate that the object has DRM protection.

K.3 OMA DRM Vendor Extension

The MTP specification [MTP] defines a vendor-extension mechanism, which enables adding additional functionality to the protocol. A vendor-extension is defined in this Appendix. This extension provides a mechanism for 2-ways transfer OMA DRM 2.x protected content via MTP, while preserving MTP functionality such as metadata-based enumeration, 2-wire plays from device and content playback capability description with hosts which do not support OMA DRM 2.x.

This extension consists of no new operations, no new device properties, 2 new object properties, no new responses and no new events. It defines a mechanism by which OMA DRM 2.x (P)DCFs can be transferred in such a way as to be fully transparent to any transferring application, while accommodating the restrictions placed on it by OMA DRMv2.

The extension defines two MTP Object Properties:

- a UINT8, which identifies the OMA DRM Status of an object
- a binary object property, which contains a Rights Object for this object.

These two Object Properties should be supported by Devices that support OMA DRM 2.x and MTP.

K.3.1 Extension Identification

Devices indicate support for this MTP extension by including a pre-defined string in the Vendor Extension Description field of the MTP DeviceInfo dataset, returned as a response to the MTP GetDeviceInfo operation. For more information, refer to [MTP].

Dataset field	*Field order*	*Size (bytes)*	*Datatype*
MTPVendorExtensionDesc	4	Varies	"www.openmobilealliance.org/omadrmv2: 1.0

Table 24: OMA DRM MTP Vendor Extension Identification

K.3.2 Extension Object Properties Summary

The object properties included in the OMA DRM vendor extension are:

- *OMA_DRM_Status* which identifies whether an object is protected by OMA DRM 2.x (or not).
- *OMA_DRM_Rights_Object* which may contain a Rights Object for a specified content.

Object Property Datacode	**Object Property Name**
0xDA11	OMA DRM Status
0xDA12	OMA DRM Rights Object

Table 25: Extension Object Property Summary Table

K.3.3 OMA DRM Status Property

Field name	**Field order**	**Size (bytes)**	**Datatype**	**Value**
PropertyCode	1	2	UINT16	0xDA11
Datatype	2	2	UINT16	0x0004 (UINT16)
Get/Set	3	1	UINT8	0x01 (GET/SET)
DefaultValue	4			0x0000
GroupCode	5	4	UINT32	Device-defined
FormFlag	6	1	UINT8	0x02 Enumeration form
Description:				
This object property identifies whether an object is protected by OMA DRM 2.x. It may contain the following initial values: 0x0000: This object is not protected (clear) by OMA DRM 2.x. 0x0001: This object is protected by OMA DRM 2.x. Whenever this property contains a non-zero value, the value of the native MTP DRM_Status (0xDC9D) must also contain a non-zero value, indicating to initiators which do not support OMA DRM 2.0 that the object may be constrained in its use.				

Table 26: OMA DRM Status Object Property in MTP

K.3.4 OMA DRM Rights Object

Field name	Field order	Size (bytes)	Datatype	Value
PropertyCode	1	2	UINT16	0xDA12
Datatype	2	2	UINT16	0x4002 (AUINT8)
Get/Set	3	1	UINT8	0x01 (GET/SET)
DefaultValue	4			0x00000000 (Emtpy Array)
GroupCode	5	4	UINT32	Device-defined
FormFlag	6	1	UINT8	0x06 ByteArray form
Description:				
This object property is a byte array which contains a valid OMA DRM 2.x Protected Rights Object (see section 5.3.9. The Rights Object is intended to be one that grants rights to use this content. NOTE: Rights Objects may me transferred with the (P)DCF within the Mutable DRM Information Box. However, OMA DRM 2.x permits Rights Object to be stored out side of the (P)DCF on the device (e.g. a Parent RO, stored in a .oro file). This property allows transferring such Rights Object separated from the (P)DCF.				

Table 27: OMA DRM Rights Object in MTP

K.4 Additional information

K.4.1 Multiple OMA DRM Containers

In the case of DCFs containing multiple OMA DRM containers (see [DRMCF-v2 section 6.4]), the MTP object-oriented approach does not fit very well. This specification recommends that only the first container should be identified as the object to transfer via MTP. Secondary OMA DRM Containers will be transferred transparently via MTP.

K.4.2 ROAP over MTP

This specification does not define an implementation of ROAP over MTP. OMA DRM 2.x Devices are recommended to obtain rights via the normal methods defined in OMA DRM 2.x.