DRM Specification V2.0

Candidate Version 2.0 – 10 December 2004

Open Mobile Alliance

OMA-DRM-DRM-V2_0-20041210-C

CONTENTS

FIGURES

TABLES

1. Scope

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

The scope of OMA “Digital Rights Management” (DRM) 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], [DRMCF] 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 system. The specification addresses specific requirements enumerated in the Release 2 Requirements document [DRMREQ-v2].

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

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

[AES] NIST FIPS 197: Advanced Encryption Standard (AES). November 2001.
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. 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, http://www.openmobilealliance.org

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

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

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

[DRMCF-v2] DRM Content Format, OMA, v2

[DRMERELD-v2] "Enabler Release Definition for DRM V2.0". Open Mobile AllianceTM. OMA-DRM-ERELD-V2_0. http://www.openmobilealliance.org/

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

[DRMREL-v2] DRM Rights Expression Language, OMA, v2

[DRMREQ-v2] DRM Requirements Specification, OMA, v2

[HMAC] RFC 2104: HMAC: Keyed-Hashing for Message Authentication. H. Krawczyk, M. Bellare, and R. Canetti. Informational, February 1997. 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. http://www.ietf.org/rfc/rfc2616.txt

[IOPPROC] "OMA Interoperability Policy and Process", Version 1.1, Open Mobile Alliance(tm), OMA-IOP-Process-V1_1, 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.
http://www.ietf.org/rfc/rfc2560.txt

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

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

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

[RFC2045] "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed & N. Borenstein, November 1996, http://www.ietf.org/rfc/rfc2045.txt

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

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

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

[RFC3280] Housley, R., Polk, W., Ford, W. and D. Solo, "Internet Public Key Infrastructure - Certificate and Certificate Revocation List (CRL) Profile", April 2002. 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. http://www.ietf.org/rfc/rfc3546.txt

[SHA-1] NIST FIPS 180-2: Secure Hash Standard. August 2002.

http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

[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 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. http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

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

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

XML Schema Part 2: Datatypes. P. Biron and A. Malhotra. W3C Recommendation, May 2001.

http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

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

2.2 Informative References

[3GPP PSS] Transparent end-to-end packet switched streaming service (PSS); 3GPP TS-26.234; Protocols and codecs – Release 6. 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. 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. www.openmobilealliance.org/documents.html

[DRMARCH-v2] "OMA DRM Architecture", Open Mobile Alliance™. OMA-DRM-ARCH-V2_0. www.openmobilealliance.org/documents.html

[PUSHOTA] "Push OTA Protocol Specification." Open Mobile Alliance™. WAP-235-PushOTA. www.openmobilealliance.org/wapdownload.html

[UICC] "Smart cards; UICC-Terminal interface; Physical and logical characteristics (release 5)", ETSI

102.221 , http://www.etsi.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, http://www.3gpp.org/

[WSP] "Wireless Session Protocol Specification" Open Mobile Alliance™. WAP-230-WSP. http://www.openmobilealliance.org/wapdownload.html

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 Protected Content and Rights Object. The Rights Object and Protected 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 unauthorized 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 Protected 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.
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 Identifier	A unique string identifier of the Domain Key
Domain Key	A 128 bit symmetric cipher key
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 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 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 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.
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 unauthorized 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 Obje*ct*.
Permission	Actual usages or activities allowed (by the Rights Issuer) over Protected Content (From [ODRL 1.1])
Play	To create a transient, perceivable rendition of a resource (From [MPEG21 RDD])
Protected Content	Media Objects that are consumed according to a set of Permissions in a Rights Object.
Restore	Transferring the Protected 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.
Rights Issuer	An entity that issues Rights Objects to OMA DRM Conformant Devices.
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 Object	A collection of Permissions and other attributes which are linked to Protected 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.
ROAP Trigger	An XML document including a URL that, when received by the Device, initiates the ROAP.
Separate Delivery	A Release 1 term defined in [DRM].
Stateless Rights	Stateless Rights are Rights Objects for which the Device does not have to maintain state information.
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 or permissions is considered Stateful Rights :<interval>, <count>, <timed-count>, <datetime>, <accumulated> or <export> .
Superdistribution	A mechanism that (1) allows a User to distribute Protected Content to other Devices through potentially insecure channels and (2) enables the User of that Device to obtain a Rights Object for the superdistributed Protected 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.

3.3 Abbreviations

3GPP	3rd Generation Partnership Project
3GPP PSS	3rd Generation Partnership Project Packet-switched Streaming Service
CA	Certification Authority
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
ISO	International Standards Organization
IMSI	International Mobile Subscriber Identity
LAN	Local Area Network
ME	Mobile Equipment
MMS	Multimedia Messaging Service
MPEG	Moving Picture Expert Group
OMA	Open Mobile Alliance
OMNA	Open Mobile Naming Authority (see http://www.openmobilealliance.org/tech/omna/index.htm)
OCSP	Online Certificate Status Protocol
OTA	Over The Air (i.e. transfer over a wireless connection)
PC	Personal Computer
PDA	Personal Digital Assistant
PDCF	Packetized DRM Content Format
PDU	Protocol Data Unit
PKI	Public Key Infrastructure
PKC	Public Key Certificate
PKC-ID	PKC Identifier: the hash of the Public Key Certificate
PSS	Packet-Switched Streaming Service (see [3GPP PSS])
REL	Rights Expression Language
REK	Rights Encryption Key
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	Synchronized Multimedia Integration Language
USIM	Universal Subscriber Identity Module
SMS	Short Messaging Service
TLS	Transport Layer Security
UA	User Agent
URI	Uniform Resource Indicator
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
WIM	Wireless Identity Module
WLAN	Wireless Local Area Network

4. Introduction

There is a growing need for a rights management system in the mobile industry so that the operators and content providers can make digital content available to consumers in a controlled manner. Digital Rights Management is a set of technologies that provide the means to control the distribution and consumption of the digital media objects. OMA has already published release 1 of the DRM specifications. The release 1 specifications provide some fundamental building blocks for a DRM system. But, they lack 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. This specification addresses these missing aspects of the OMA DRM.

The OMA DRM system enables Content Issuers to distribute Protected Content and Rights Issuers to issue Rights Objects for the Protected Content. The 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. For User consumption of the Content, Users acquire Permissions to Protected Content by contacting Rights Issuers. Rights Issuers grant appropriate Permissions for the Protected Content to User Devices. The Content is cryptographically protected when distributed; hence, Protected Content will not be usable without an associated Rights Object issued for the User's Device.

The Protected Content can be delivered to the Device by any means (over the air, LAN/WLAN, local connectivity, removable media, etc.). But the Rights Objects are tightly controlled and distributed by the Rights Issuer in a controlled manner. The Protected Content and Rights Objects can be delivered to the Device together, or 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]. For a detailed discussion of the Rights Expression Language that is used to construct the Rights Objects, please refer to [DRMREL-2]. The DRM Content Format is specified in the [DRMDCF-2] specification.

This specification defines an end-to-end system for Protected Content distribution. Section 5 specifies a Rights Object Acquisition Protocol (ROAP). Section 7 describes the key management schemes utilized in this specification. Section 8 describes the Domains functionality – sharing of content and rights among a set of Devices enrolled into a Domain. Section 9 through 15 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.

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 and two protocols by which the Device requests and acquires Rights Objects (RO). 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 ROAP suite also includes 2-pass protocols for Devices joining and leaving a Domain; the Join Domain protocol and the Leave Domain protocol.

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 synchronization.

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.

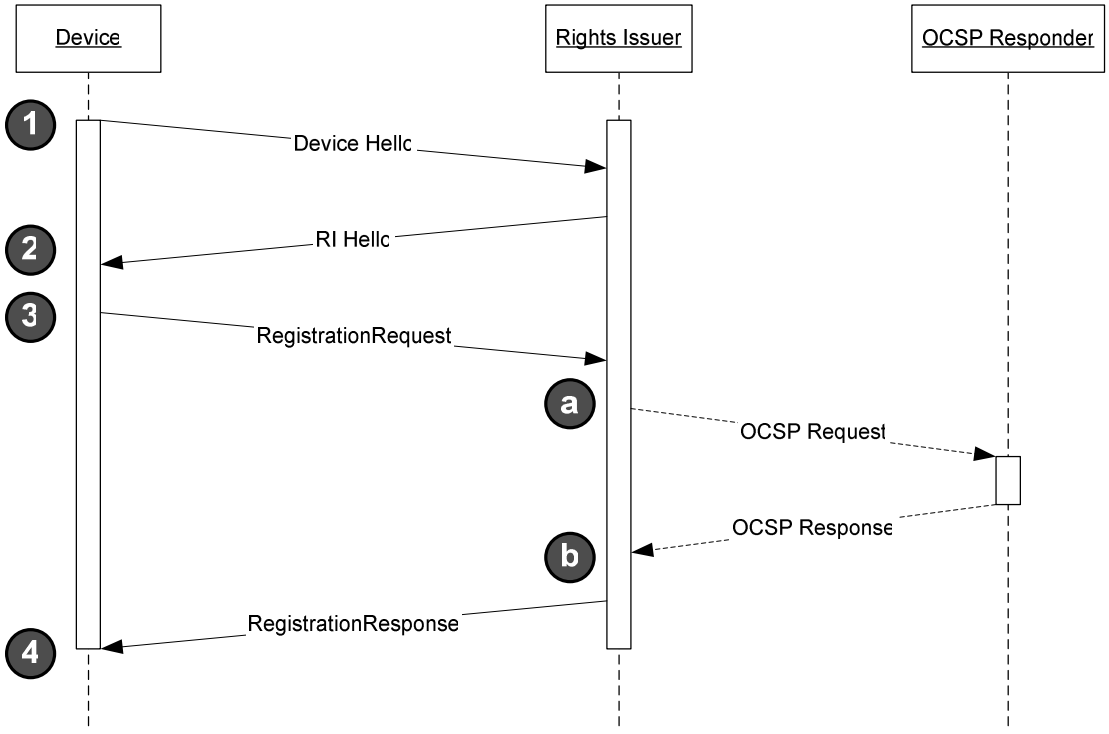

Figure 1: The 4-pass Registration Protocol

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. If the Device is an Unconnected Device that does not support DRM Time, the RI must 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 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.

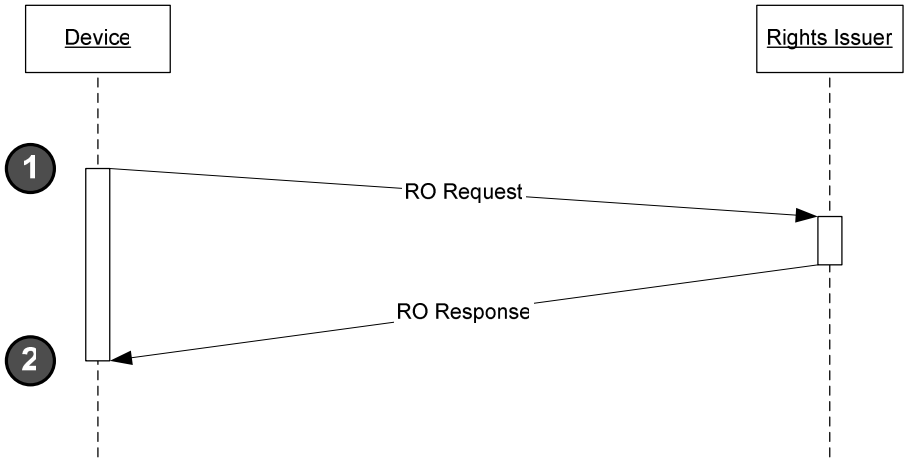

Figure 2: The 2-pass Rights Object Acquisition Protocol

5.1.3 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.

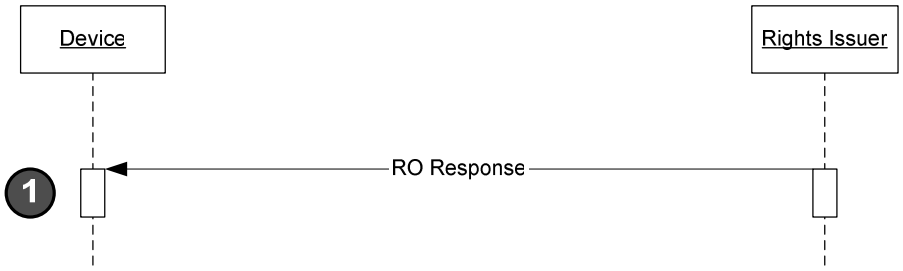

Figure 3: The 1-pass Rights Object Acquisition Protocol

5.1.4 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.

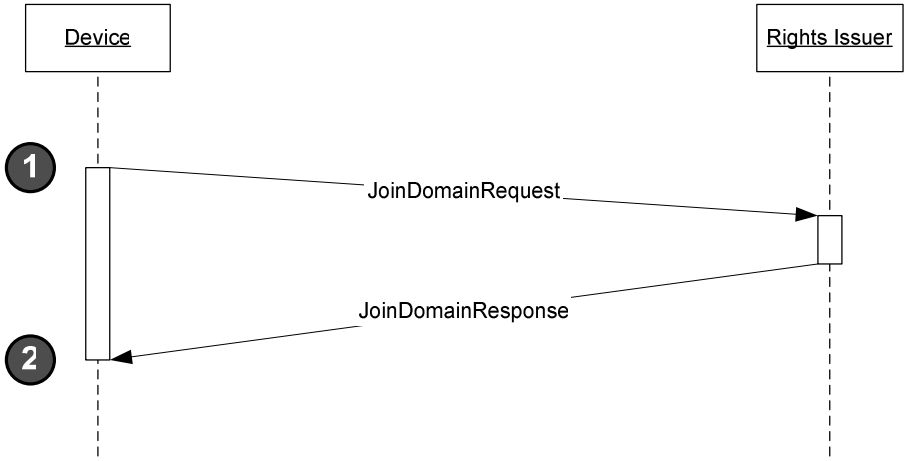

Figure 4: The 2-pass Join Domain Protocol

5.1.5 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.

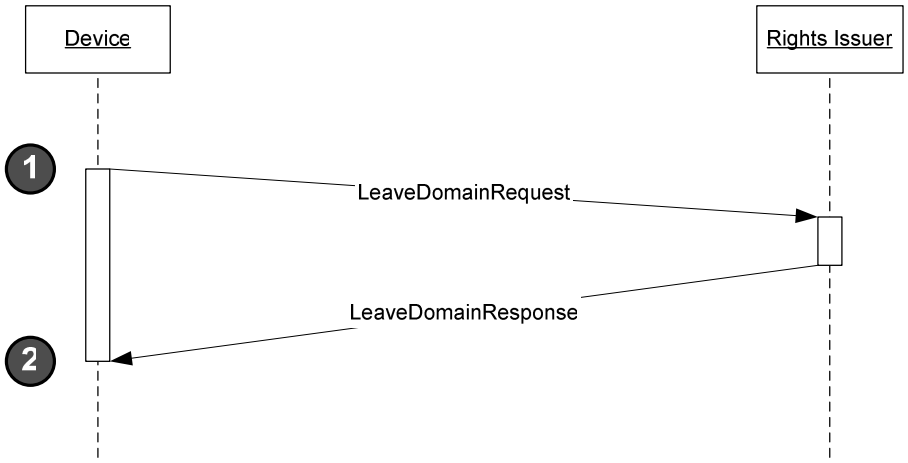

Figure 5: The 2-pass Leave Domain Protocol

5.1.6 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. Appropriate user consent MUST have been obtained prior to initiating any ROAP protocols as a result of a ROAP Trigger. Since the ROAP comprises several protocols, the ROAP Trigger includes an indication of the actual protocol (Registration, RO acquisition, Join Domain, or Leave Domain) to be started by the Device.

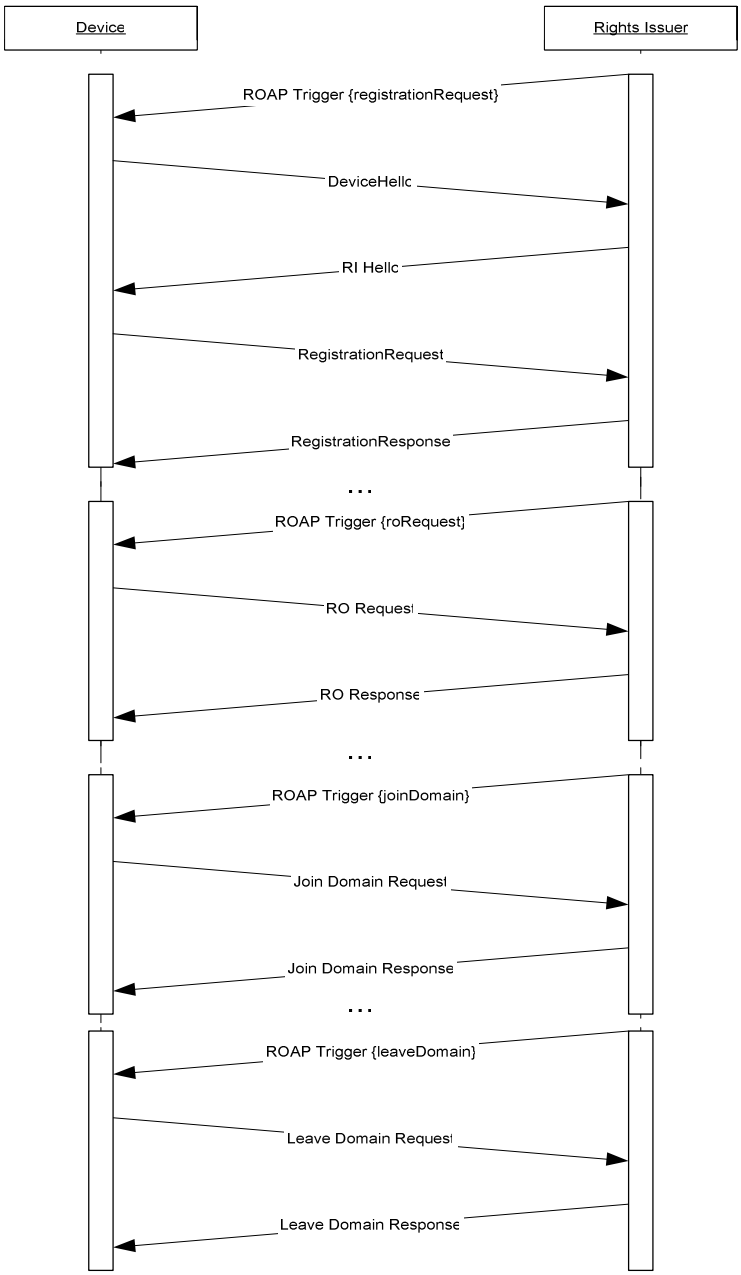

Figure 6: ROAP Trigger

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.

```
<schema
 targetNamespace="urn:oma:bac:dldrm:roap-trigger-1.0"
 xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:roap-trigger="urn:oma:bac:dldrm:roap-trigger-1.0"
 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#"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified">

<import namespace="urn:oma:bac:dldrm:roap-1.0" schemaLocation="roap.xsd"/>

<import namespace="http://www.w3.org/2000/09/xmldsig#"
      schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-
schema.xsd"/>

<import namespace="http://www.w3.org/2001/04/xmlenc#"
      schemaLocation="http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd"/>

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

<complexType name="ROAcquisitionTrigger">
 <sequence>
  <element name="riID" type="roap:Identifier"/>
  <element name="nonce" type="roap:Nonce" minOccurs="0"/>
  <element name="roapURL" type="anyURI"/>
  <element name="domainID" type="roap:DomainIdentifier"
        minOccurs="0"/>
  <sequence maxOccurs=”unbounded”>
   <element name="roID" type="ID"/>
   <element name="contentID" type="anyURI" maxOccurs="unbounded"/>
  </sequence>
 </sequence>
 <attribute name="id" type="ID"/>
</complexType>

<complexType name="DomainTrigger">
 <sequence>
  <element name="riID" type="roap:Identifier"/>
  <element name="nonce" type="roap:Nonce" minOccurs="0"/>
  <element name="roapURL" type="anyURI"/>
  <element name="domainID" type="roap:DomainIdentifier"/>
```

```
  </sequence>
  <attribute name="id" type="ID"/>
</complexType>

<!-- ROAP trigger -->
<element name="roapTrigger" type="roap-trigger: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-trigger:RegistrationRequestTrigger"/>
      <element name="roAcquisition" type="roap-trigger:ROAcquisitionTrigger"/>
      <element name="joinDomain" type="roap-trigger:DomainTrigger"/>
      <element name="leaveDomain" type="roap-trigger:DomainTrigger"/>
    </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>
</schema>
```

The **<riID>** element MUST uniquely identify the Rights Issuer. For triggers besides the **<registrationRequest>**, 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.

The **<nonce>** element provides a way to couple ROAP triggers with ROAP requests. When present, a Device MUST include the **<nonce>** value as a ***nonceTrigger*** attribute in a subsequent ROAP request. See further Section 5.3.4 below. RIs MUST include a **<nonce>** element in "LeaveDomain" triggers. RIs MUST follow the guidelines for nonces as expressed in Section 5.3.10 below.

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 as described below to determine which ROAP PDU to send:

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

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.

One or several **<contentID>** elements MUST be included in the **<roAcquisition>** trigger, at least one for each **<roID>** element. In the case where a single RO applies to several DCFs the <roAcquisition> trigger MUST include a <contentID> element for each DCF. In the case where a single RO applies to several Content Objects inside one DCF (Multipart DCF), the trigger SHALL include the <contentID> element of the first Content Object in the Multipart DCF. The **<contentID>** elements MUST contain the ContentID as specified in the *ContentID* field in the Common Header of the Content Object inside the associated DCF (or PDCF) [DRMCF-v2].

If the DRM Agent has a valid RI Context with the identified Rights Issuer, then the DRM Agent MAY initiate the ROAP transaction without acquiring consent from the user. If the DRM Agent does not have a valid RI Context with the identified Rights Issuer, then the DRM Agent MUST obtain user consent before connecting to the RI and initiating the transaction.

In the case where a DRM Agent receives a ROAP Trigger where the **<roapTrigger>** element carries a **<registrationRequest>** element, the DRM Agent MUST use the value of the **<riID>** element to verify that it has a valid RI Context with the Rights Issuer, prior to initiating the transaction.

The Rights Issuer MAY authenticate the ROAP Trigger by including a signature of the trigger in the **<signature>** element (see section 5.3.3 for information on digital signatures). The RI MUST include a **<signature>** element if a **<leaveDomain>** element is present. With one exception (see below), Devices MUST verify signed ROAP triggers. 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 trigger is a "LeaveDomain" trigger, the Device is not a member of the identified Domain, and the trigger has been authenticated with a MAC based on the Domain Key. In this case, the Device MUST acquire user consent before initiating the ROAP.

The **<ds:Reference>** element of the **<ds:SignedInfo>** child element of the **<signature>** shall reference a **ROAPTrigger** element by using the same value for the **URI** attribute as the value for the ROAP trigger element's ***id*** attribute. In the case of a “LeaveDomain” trigger, 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".

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 "**1.0**". 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 14) MUST start the procedures specified in section 11.6.4. 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”.

5.2.2 Initiating ROAP from a DCF

This section applies ONLY to Connected Devices.

If the DRM Agent receives a DCF with both a Silent header and a Preview header, the DRM Agent MUST give priority to the header that appears first in the DCF.

If the DRM Agent has a valid RI Context with the RI, the DRM Agent MAY attempt to acquire Rights silently for the DCF 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, In this case, the DRM Agent MUST compare the domain name of the silent or preview URL with the list of authorized 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

authorized domain names already stored by the DRM Agent for that RI, the DRM Agent MUST silently attempt to acquire the RO for the DCF by sending a request message to the silent or preview URL stored in the DCF, and responding to the ROAP Trigger and/or Download Descriptor that will be returned by the Rights Issuer.

The RI MUST return a suitable ROAP error if this RO request cannot be reconciled to a prior purchase transaction. Upon receipt of a ROAP error, the Device MAY take further action. In this case, if the context is a user-initiated session, it is recommended that the Device start a browsing session with the RI by sending a request to the DCF *RightsIssuerURL*. If the context is a DRM Agent-initiated session to acquire rights silently and automatically, then the DRM Agent is RECOMMENDED to abandon the rights acquisition effort.

In all other cases, the DRM Agent MUST NOT attempt to silently acquire the RO for the DCF. It MUST obtain the user's consent before attempting to acquire an RO for the DCF. Once the user has given consent, the DRM Agent MUST send a request to the DCF *RightsIssuerURL,* and MUST be prepared to receive either an XHTML page or ROAP Trigger from the RI. The DRM Agent MUST NOT attempt to acquire an RO for the DCF if the user does not provide consent. The DRM Agent MAY store the DCF, however, even if the user does not give consent for RO acquisition.

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 DCF, the DRM Agent MUST add the domain name of the silent or preview URL to the list of authorized domain names for that RI, if the domain name is not already present. As specified in section 5.4.2.4, 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.

5.3 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 G.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, normalization 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.

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 Canonicalization & 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). The input to the digital signature operations and the MAC operations SHALL be the canonical form of XML data in accordance with [XC14N]. DRM Agents and RIs MUST send integrity-protected information in canonicalized form and MUST NOT employ any subsequent transformations or modifications to such content. Despite this, DRM Agents SHOULD, and RIs

MUST, canonicalize received and integrity protected information before verifying digital signatures and MACs calculated on the information.

Note that all ROAP XML PDUs are XML 1.0 data.

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. All ROAP requests MAY contain a ***triggerNonce*** attribute. The ***triggerNonce*** attribute MUST be present in a ROAP request if and only if a ROAP trigger containing a **<nonce>** element initiated the ROAP request. In this case, the value of the ***triggerNonce*** attribute MUST be the same as the value of the ROAP trigger's **<nonce>** element.

```
<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:Status" use="required"/>
</complexType >
```

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.

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

Upon transmission or receipt of a message for which Status is not "Success", the default behaviour, unless explicitly stated otherwise below, is that both the RI and the Device SHALL immediately close the connection and terminate the protocol. RI systems and Devices are required to delete any session-identifiers, nonces, keys, and/or secrets associated with a failed run of the ROAP protocol.

When possible, the Device SHOULD present an appropriate error message to the user.

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 authorized 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 recognized 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.

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, and ROAP-LeaveDomainResponse.

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, and ROAP-LeaveDomainResponse.

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, and ROAP-LeaveDomainResponse.

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, and ROAP-LeaveDomainResponse.

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, and ROAP-LeaveDomainResponse. The Device SHOULD initiate the 4-pass Registration protocol, using the ***riURL*** as stored in the RI Context. The Device MUST have user consent to contact the RI, in the same way as for processing ROAP triggers as specified in section 5.2.1.

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-ROResponse, ROAP-JoinDomainResponse, and ROAP-LeaveDomainResponse. The Device SHOULD initiate the 4-pass Registration protocol, using the ***riURL*** as stored in the RI Context. The Device MUST have user consent to contact the RI, in the same way as for processing ROAP triggers as specified in section 5.2.1.

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 unrecognized 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.

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 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. The ProtectedRO type is used to carry protected Rights Objects in ROAP-ROResponse messages and Domain ROs (when sent in DCFs or separately).

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

<complexType name="ProtectedRO">
  <sequence>
    <element name="ro" type="roap:ROPayload"/>
    <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. The MAC is calculated over the complete **<ro>** element. Before performing the MAC calculation, the Rights Issuer MUST canonicalize the **<ro>** element in accordance with [XC14N]. See also Section 5.3.3. 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.

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".

5.3.9 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"/>
  </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]. The ***o-ex:id*** attribute of this type SHALL be present.

The **<signature>** element is of type **ds:SignatureType** from [XMLDsig] and MUST be present when the RO is a Domain RO. 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). Before performing the signature calculation, the Rights Issuer MUST canonicalize all elements the signature shall be made over, in accordance with [XC14N]. See also Section 5.3.3. 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 9.4.

The **<encKey>** element is of type **xenc:EncryptedKeyType** from [XMLEnc]. It consists of a wrapped concatenation of a MAC key, K_{MAC} and an RO encryption key, K_{REK}. The ***Id*** attribute of this element SHALL be present and SHALL have the same value as the value of the ***URI*** attribute of the **<ds:RetrievalMethod>** element in any **<ds:KeyInfo>** elements inside the **<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 of type **roap:X509SPKIHash**, identifying a particular DRM Agent's public key through the (SHA-1) hash of the DER-encoded subjectPublicKeyInfo value in its certificate. In the case of a Rights Object intended for a Domain, it will be of the type **roap:DomainIdentifier**, identifying the correct Domain key. Note that the encrypted key material consists of ***two*** keys - a MAC key and a Rights Object Encryption key. For further information, see the Key Management discussion in section 7.

The ***version*** attribute indicates the version of the ROPayload type. For this version of the OMA DRM specification, the value SHALL be “1.0”. 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 will, when applicable, correspond to an **<roID>** value in a previous ROAP-RORequest. The ***id*** attribute is also used as a reference point for the MAC as described in the previous section.

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, for example, to register with the RI or to join a Domain (as specified in section 8.6.2.1) indicated in a **roap:DomainIndentifier** element. The value of the ***riURL*** MUST be a URL according to [RFC2396], and MUST be an absolute identifier. If

the Device utilizes the ***riURL*** for registration purposes, the Device MUST have user consent to contact the RI, in the same way as the requirement for processing ROAP triggers as specified in section 5.2.1.

5.3.10 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 Base64-encoded characters long (approx. 80 bits). Devices MUST at least support nonce values 14 Base-64 encoded characters 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 three categories: Registration, RO Acquisition, and Domain management.

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 1: 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 "**1.0**". 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. 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

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.

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: The following extensions are defined for the ROAP-DeviceHello message:

Certificate Caching: The presence of this extension indicates to the RI that the Device has a capability to store information in the RI context whether an RI has stored Device certificate information or not. (Note: This is not about whether the Device has stored RI certificate information or not. For this, the *Peer Key Identifier* extension is used - see the ROAP-RegistrationRequest, ROAP-RORequest, and ROAP-JoinDomainRequest messages.) If the Device has this capability, then the Device MUST include the *Certificate Caching* extension. If this extension is used, the RI can use the *Peer Key Identifier* or the *Certificate Caching* extension in its ROAP-RIHello message to indicate that it has stored the Device public key or that it is capable of storing Device certificate information, respectively.

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 "**1.0**".

```
<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"/>
```

The following extension is defined for the ROAP-DeviceHello message:

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

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 2: 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. Other identifiers are allowed but interoperability when using them is not guaranteed. 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 5.3.10.

Trusted Device Authorities is a list of Device trust anchors recognized 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 recognized 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 extension is empty or if the identifier matches the Device's current public key, it means the RI has already stored the Device ID and the corresponding Device certificate chain, and the Device need not send its certificate chain in a later request 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 *Certificate Caching* extension was present in the ROAP-DeviceHello message and 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 *Certificate Caching* extension was not present in the ROAP-DeviceHello message, then the RI need not send the *Certificate Caching* extension in its ROAP-RIHello. 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="hexBinary"/>
  </extension>
 </complexContent>
</complexType>

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

The following schema fragment defines the *Peer Key Identifier* extension:

```
<complexType name="PeerKeyIdentifier">
 <complexContent>
```

```
   <extension base="roap:Extension"/>
      <sequence minOccurs=”0”>
       <element name="identifier" type="roap:KeyIdentifier"/>
      </sequence>
   </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="string"/>
       <element name="model" type="string"/>
       <element name="version" type="string"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

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 3: 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 5.3.10.

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**”.

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 recognized 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's current public key, or if the extension is empty, 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. 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 canonicalized using the exclusive canonicalization method defined in [XC14N].

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="hexBinary" use="required"/>
   </extension>
  </complexContent>
</complexType>
```

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 4: 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 URL that SHOULD be stored in the RI Context. This URL SHOULD be used by the Device in later interactions with the RI to send ROAP requests. 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].

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

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 along 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 and the current one except the *Signature* element is canonicalized using the exclusive canonicalization method defined in [XC14N].
- 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: riURL, RI ID, Selected Version, Selected Algorithms, and a Certificate Caching indication if the RI has stored the Device certificate or not (all this information is carried in the ROAP-RIHello message). 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. 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 riURL stored in the RI Context. 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.

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="hexBinary"/>
    </extension>
  </complexContent>
</complexType>
```

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="string"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

5.4.3 RO Acquisition

5.4.3.1 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.

5.4.3.1.1 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 5: RO Request Message Parameters

Device ID identifies the requesting Device as specified in section 5.4.2.1.1.

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

RI ID identifies the authorizing RI as specified in section 5.4.2.2.1.

Device Nonce is a nonce chosen by the Device. Nonces are generated and used in this message as specified in section 5.3.10.

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 [DRMCF-v2] using the SHA-1 algorithm.

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

The Device MUST send the *Peer Key Identifier* extension if, and only if, it has stored the RI public key. The Device MUST send the *No OCSP Response* extension if, and only if, it has a complete set of valid OCSP

responses for the RI certificate chain. The Device MUST send the *OCSP Responder Key Identifier* extension if, and only if, it has stored an OCSP Responder key for this RI.

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

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

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

5.4.3.1.2 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>
```

5.4.3.2 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.

5.4.3.2.1 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 6: 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 RI selects the Device ID of the recipient Device. 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 be an RI identifier as specified in section 5.4.2.2.1.

Device Nonce: This parameter, if present (2-pass), MUST have the same value as the corresponding parameter value in the preceding ROAP-RORequest.

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

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 a preceding ROAP-RegistrationRequest (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 9.7. 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.

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

The Device MUST verify this signature. A Device MUST NOT accept the 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.

5.4.3.2.2 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 The Protected Rights Object payload type section.

5.4.4 Domain Management

5.4.4.1 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.

5.4.4.1.1 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 7: Join Domain Request Message Parameters

Device ID identifies the requesting Device as specified in section 5.4.2.1.1.

RI ID identifies the authorizing RI as specified in section 5.4.2.2.1.

Device Nonce is a nonce chosen by the Device. Nonces are generated and used in this message as specified in section 5.3.10.

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 RI's current public key, 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 8.7.1.

The Device MUST send the *Peer Key Identifier* extension if, and only if, it has stored the RI public key. The Device MUST send the *No OCSP Response* extension if, and only if, it has a complete set of valid OCSP responses for the RI'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 canonicalized using the exclusive canonicalization method defined in [XC14N].
- The result of the canonicalization, *d*, is considered as input to the signature operation.
- The signature is calculated on *d* in accordance with the rules of the negotiated signature algorithm.

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

5.4.4.1.2 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 8.7 for further information). 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>
```

5.4.4.2 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.

5.4.4.2.1 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 8: 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.

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. 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). 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 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 canonicalized using the exclusive canonicalization method defined in [XC14N].
- The result of the canonicalization, *d*, is considered as input to the signature operation
- The signature is calculated on *d* in accordance with the rules of the negotiated signature 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, the riURL (as stored in the associated RI Context), and, if applicable, an indication that the RI supports hash-chained Domain Keys. 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. The Domain Context SHALL also contain the RI Public Key for the case when the Domain Context Expiry Time extends beyond the RI Context Expiry Time.

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.

5.4.4.2.2 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:

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

<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 **<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 of type **roap:X509SPKIHash**, 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 [XC14N] version 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.

5.4.4.3 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.

5.4.4.3.1 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 9: Leave Domain Request Message Parameters

Device ID identifies the requesting Device as defined in section 5.4.2.1.1.

RI ID identifies the authorizing RI as defined in section 5.4.2.2.1.

Device Nonce is a nonce chosen by the Device. Nonces are generated and used in this message as specified in section 5.3.10.

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

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

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

5.4.4.3.2 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>
```

5.4.4.4 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.

5.4.4.4.1 Message description

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

Table 10: 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.

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).

5.4.4.4.2 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>
```

6. Certificate Status Checking & Device Time Synchronization

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 the Device certificate.

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 and status of the RI certificate and any other revocable certificates in the RI certificate chain. To allow the Device to do the certificate status check, the RI MUST include a complete set of OCSP responses for its 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.

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.

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 D).

DRM Agents MUST support all client requirements in [OMA-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 Synchronization

An RI, which receives a ROAP-RORequest or a ROAP-JoinDomainRequest, 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.

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 adjust its time to match the time in the `producedAt` component of the OCSP response, assuming the

Registration protocol exchange otherwise was successful. 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,* 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:

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

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:

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

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

$$c_1 = \text{RSA.ENCRYPT}(P,Z) = Z^e \bmod m$$

$$C_1 = I2OSP(c_1, mLen)$$

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* | Hash (*Z* | *D* | *otherInfo*).

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

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

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.:

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

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

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

$$C = C_1 \mid C_2$$

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*:

$$C_1 \mid C_2 = C$$

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

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

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

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

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

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

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

7.3 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, K_M. The RI then applies KDF on K_M at least as many times *n* as the RI believes there will be generations of the Domain:

$DK_n = K_M$

DK_{n-1} = KDF(DK_n, NULL, *kekLen*)

DK_{n-2} = KDF(DK_{n-1}, NULL, *kekLen*)

…

DK_0 = KDF(DK_1, 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 5.4.4.2 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} = KDF(DK_i, NULL, 16)

DK_{i-2} = KDF(DK_{i-1}, 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 5.4.4.1.1 and 5.4.4.2.1, 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 5.4.4) 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

To be able to use a Domain RO, a Device must have joined the corresponding Domain.

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.

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 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 5.4.4) to leave a Domain. A Device may do this by sending a LeaveDomainRequest message to the riURL as stored in the Domain Context or as a result of receiving a **<leaveDomain>** ROAP Trigger.

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

8.5 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.

8.6 Domain RO Processing Rules

8.6.1 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.

8.6.2 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 8.6.2.1.

8.6.2.1 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 MAY keep the Domain RO and SHOULD register with the RI by sending a ROAP-DeviceHello request to the URL specified in the ***riURL*** attribute of the **roap:ROPayload.** The Device MUST acquire the user's consent prior to initiating the protocol.

The Device MUST verify the signature of the Domain RO using the RI's Public Key. If the verification fails the Device MAY request a new Rights Object by sending a request to the RightsIssuerURL in the relevant DCF.

After the Device verifies the signature of the Domain RO, it SHOULD compare the **<domainId>** field within the Domain RO with the Domain identifiers for any 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 stored domain identifier. The Device MAY install the Domain RO.

2. The **<domainId>** field matches the first 17 digits of a stored domain identifier, but the Domain Generation of the RO is greater than the Generation of the stored domainId. The device MAY attempt to upgrade the Domain by sending a ROAP-JoinDomainRequest to the URL specified in the ***riURL*** attribute of the **roap:ROPayload** (see section 5.3.9). The Device SHOULD perform this action silently, if the user has given permission for silent communication with this RI. If the user has not given permission for silent communication, the Device MUST acquire the user's consent prior to sending the request, but SHOULD present messages to the user indicating that

the request is for Domain upgrade and not for joining an entirely new Domain. If the Domain upgrade is successful, the Device MAY install the Domain RO.

3. The **<domainId>** field does not match a stored domain identifier. The Device MAY attempt to join the Domain by sending a ROAP-JoinDomainRequest to the URL specified in the ***riURL*** attribute of the **roap:ROPayload**. The Device MUST acquire the user's consent prior to sending a request. If the JoinDomain protocol is successful, the Device MAY install the Domain RO.

When an RI receives a ROAP-JoinDomainRequest, it will decide (using its own criteria) whether to allow the Device to join the Domain or not. If the RI chooses to allow the Device to join the Domain then the RI MUST return a successful ROAP-JoinDomainResponse to the Device. In the event that the RI chooses not to allow the Device to join the Domain, the Device MAY request a Device RO by sending a request to the RightsIssuerURL in the relevant DCF.

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 MAY request another RO by sending a ROAP-RORequest to the URL specified in the ***riURL*** attribute of the **roap:ROPayload**.

If the Domain RO is stateful**,** then the Device MUST perform the replay protection related checks defined in Section 9.4.

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 MUST remove the Domain RO from the DCF and discard it. The Device MAY request a valid RO for the DCF by sending a request to the RightsIssuerURL in the DCF.

8.6.2.2 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 [DRMCF-v2] 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.

8.7 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 5.4.4). 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 domainId 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 **<domainID>** field matches the first 17 digits of a stored domainId, 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, if the user has given permission for silent communication with this RI and if the trigger was authenticated. If the user has not given permission for silent communication, or if the trigger was not authenticated, the Device MUST request user permission to upgrade the Domain, but SHOULD present appropriate messages to the user indicating that the request is for Domain upgrade and not for joining an entirely new Domain.
2. If the **<domainID>** field does not match a stored domainId, then the Device is not a member of the Domain. The Device MUST behave as if it had received a domain-RO for a Domain it was not a member of, as specified in section 8.6.2.1.

8.7.1 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.

Example1. 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. Protection of Content and Rights

9.1 Protection of Content Objects

The Content Objects are protected by symmetric key encryption. The details of the content format are specified in [DRMCF-v2] 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 Protected Content.

For integrity protection of the DCF, a cryptographic hash value of the DCF is generated and inserted into the Rights Object. This hash value MUST be generated according to the DCF hash calculation procedure specified in section 12.4. DRM Agents in client Devices MUST verify that the hash value in the Rights Object 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. 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.

9.2 Composite Content Objects and Associated Rights Objects

9.2.1 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.

9.2.1.1 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 realized 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]).

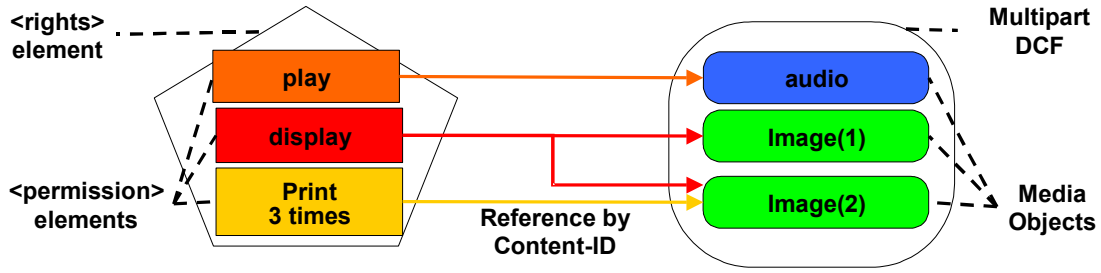

Figure 7: Multiple Rights for Multipart DCFs

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).

9.3 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-authorized modification of the rights information within the Rights Object. The syntax and semantics of the Rights Object is specified in the [DRMREL-v2] 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.

If a Rights Object is associated with a Multipart DCF, all the rights expressions for the component elements MUST be expressed within a single **<rights>** element. Rights Objects associated with Multipart DCFs may also contain a number of protected CEKs, since each element may be protected with a unique CEK.

The Rights Object MUST be assigned a unique identifier by the Rights Issuer.

9.3.1 Device RO Processing Rules

9.3.1.1 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 8.6.

9.3.1.2 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 9.3.1.3.

9.3.1.3 Installing a Device RO

When a Device receives a Device RO through a successful execution of the RO Acquisition protocol, it MUST proceed as follows:

- 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 SHOULD 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 **<riID>** element in the **roap:ROPayload** of a Device RO does not match the RI Identifier in any valid RI context, the Device MAY attempt to register with that RI by initiating the Registration protocol. The Registration protocol is initiated by sending a ROAP-DeviceHello request to the URL specified in the ***riURL*** attribute of the **roap:ROPayload**. The Device MUST acquire the user's consent prior to initiating the protocol. 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 9.4.

If the RI Context has expired (indicated by the RI Context Expiry Time) the Device MUST NOT install ROs for this RI and the Device SHOULD delete the RI Context.

If the Device RO is received within a DCF and if the Device cannot verify the signature of the Device RO, the Device MUST remove the RO from the DCF and discard it. The Device MAY request a valid Device RO for the associated DCF by sending an HTTP request to the RightsIssuerURL in the DCF.

9.4 Replay Protection of Stateful Rights Objects

9.4.1 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 unauthorized extension of the permissions.

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 synchronization 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.

9.4.2 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 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 ROs with GUIDs and RITS, and one with <GUID> entries for ROs with GUIDs only. The Device MUST protect the integrity of its replay caches. It is RECOMMENDED that each replay cache is able to store at least 100 entries.

9.4.2.1 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 synchronized time across the set of RIs and OCSP responders that may be accessed by a Device.

When receiving a stateful RO with a **<timestamp>** element (RITS), the Device MUST perform the following procedure:

a) 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 synchronization 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.

9.4.2.2 Stateful ROs without RI Time Stamps

This replay protection mechanism is mainly 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 9.4.2.1 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.

9.5 Parent Rights Object

A Rights Object may inherit Permissions from another Rights Object, using the <inherit> syntax as specified in [DRMREL-v2]. 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.

9.5.1 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.

9.5.2 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 9.5 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.

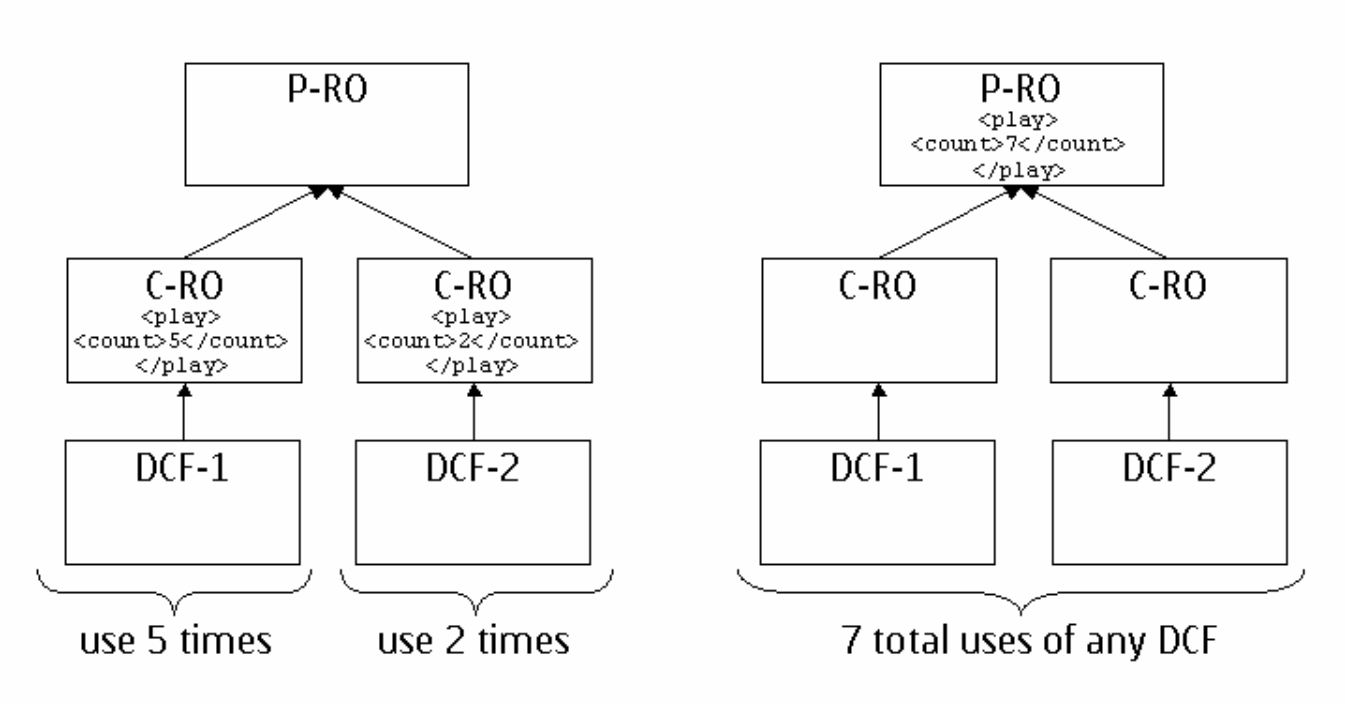

Figure 8: Parent ROs and Associated Semantics

9.6 Off-Device Storage of Content and Rights Objects

Because Devices have a limited amount of storage space in which to store Protected Content and Rights Objects, users may desire to move Protected Content and Rights Objects off the Device, e.g. to removable memory, a personal computer, or a network store to make room for new Protected 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 Protected Content and Rights Objects from the remote storage back onto the Device store.

As explained in earlier sections of this specification, both the Protected Content and Rights Objects are protected and bound to a specific Device or a Domain. For this reason, Protected Content and Rights Objects MAY be allowed to leave the Device provided the following condition is met:

- The Protected 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.

9.7 Group ID Mechanism

A content object (DCF) MAY contain an *OMADRMGroupID* Box that defines the group identity of the DCF, as specified in [DRMCF-v2].

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 5.9 of [DRMREL-v2].

A Rights Object bound to a DCF Group ID MUST NOT include any DCF hash values, as described in Section 8.1. 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.Capability Signaling

10.1 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].

10.2 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)

In addition to the supported media types, Devices MUST advertise the DRM version using the "<major>.<minor>" format defined 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" ":" *DIGIT "." *DIGIT
```

10.3 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 protected content, for example, a DRM system used on a memory card. See Appendix G.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 protected 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 [DRMREL-v2]	"play", "display", "execute", "print"	Supported permissions using the same syntax as defined in the REL specification.
DrmConstraint	Optional DRM permission constraints as defined in	"datetime", "interval", "accumulated"	Supported constraints using the same syntax as

	[DRMREL] or [DRMREL-v2]		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	"Very Secure Card"	A textual name of the external DRM system. Well-defined names for external DRM systems are managed by OMNA.

Table 11: User Agent Profile Attributes

10.4 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 specs), the:

- Content Issuer MAY issue Forward Locked Content.
- Content Issuer MAY issue 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 Separate Delivery Content only if the Device advertises support for Separate Delivery.
- Content Issuer MAY issue Separate Delivery Content encapsulated in a DRM Message, only if the Device advertises support for Separate Delivery.
- Rights Issuer MAY issue a DRM v1 Rights Object for Separate Delivery Content, only if the client advertises support for Separate Delivery.
- Content Issuer SHOULD issue DRM v2 content.
- Content Issuer MAY issue DRM v2.0 Content encapsulated in a DRM Message.
- Rights Issuer SHOULD issue a DRM v2.0 Rights Object for DRM v2.0 Content.
- Rights Issuer SHOULD send the ROAP Trigger to initiate the ROAP protocol (see section 5.2.1).
- Rights Issuer MUST NOT issue any DRM v2.0 Rights Object for Combined Delivery Content, even if the Device advertises support for Combined Delivery.
- Rights Issuer MUST NOT issue any DRM v2.0 Rights Object for 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 v2.0 Content.

11.Transport Mappings

11.1 Introduction

The following sections describe how ROAP PDUs are delivered using typical delivery protocols, the most common being HTTP. Examples are provided in Appendix G.1.

A Connected Device MUST support HTTP for transporting ROAP PDUs as described in section 11.2.

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 14.

An Unconnected Device SHALL support the functionality to use connectivity provided by a Connected Device, as described in section 14.

11.2 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 G.2.

11.2.1 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].

11.2.2 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.

11.2.3 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/mixed*

Example:

Accept: application/vnd.oma.drm.roap-pdu+xml, multipart/mixed

- HTTP requests from the DRM Agent MUST NOT contain more than one ROAP request message.

11.2.4 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/mixed* (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/mixed; boundary="XX---XX"

  DRM Agents MUST NOT accept any other HTTP *Content-Type* header values in responses to ROAP requests and MUST terminate the ROAP session if such responses are received.
- 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

11.2.5 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).

11.3 OMA Download OTA

A Rights Issuer MAY use OMA Download OTA [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 installation notification as a billing trigger.

Depending on specific deployment and business scenarios, OMA Download OTA can be used in different ways in the context of delivering protected content and Rights Objects. Appendix G.3 illustrates this with the help of a few examples, but is not meant to be an exhaustive collection of deployment scenarios.

11.3.1 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.

11.3.1.1 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 ore 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.

11.3.1.2 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 *objectURI* 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.
- 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 5.1.6 . 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.

11.3.1.3 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.
- 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 5.1.6. 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.

11.4 WAP Push

11.4.1 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.

11.4.2 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.

11.5 MMS

The Multimedia Messaging Service (MMS) may be used to transfer Protected Content in MMS Protocol Data Units (PDUs) as defined in [MMSENC]. In regard to DRM two use cases need to be distinguished:

a) The Protected 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 Protected 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 G.4.

11.6 ROAP over OBEX

11.6.1 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 14) 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 G.5.

11.6.2 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
```

11.6.3 OBEX Profile

11.6.3.1 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 14) and Unconnected Devices MUST support these OBEX operations.

OBEX Operation	Opcode
Connect	0x80

OBEX Operation	Opcode
Disconnect	0x81
Put	0x02 (0x82)
Get	0x83
Abort	0xFF

11.6.3.2 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 14) 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.

11.6.3.3 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

11.6.3.4 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

11.6.3.5 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	Response code	0xA0 for success; otherwise the client will disconnect with a failure indication

Field/Header	Name	Explanation/Value
Field	Packet length	Varies

11.6.3.6 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).

11.6.3.7 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

Field/Header	Name	Explanation/Value
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 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.

11.6.4 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 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 11.6.5. 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).

11.6.4.1 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.

11.6.5 Service Discovery

11.6.5.1 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

11.6.5.2 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 12 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 13 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 13 SDP PDUs

11.6.6 Bluetooth Considerations

11.6.6.1 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 initialization procedure, i.e. be paired. The initialization 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.

12.Super Distribution

12.1 Overview

OMA DRM v2 provides two mechanisms for superdistribution:

- Protected 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 [DRMCF-v2]), this URL MAY be distributed from one Device to another without any restrictions.

12.2 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 [DRMCF-v2] for further details on the appropriate DCF headers.

12.3 Transaction Tracking

A DCF may contain a TransactionID as an information element inside an OMADRMTransactionTracking box according to [DRMCF-v2]. 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.* The DRM Agent 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 Protected 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.

12.4 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.

13.Export

13.1 Introduction

After downloading OMA DRM protected content, the User may wish to render that content on another Device that has a different DRM protection format. Export is an operation in which the DRM content and corresponding Right Object 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]) 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.

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 9 below explains the principle.

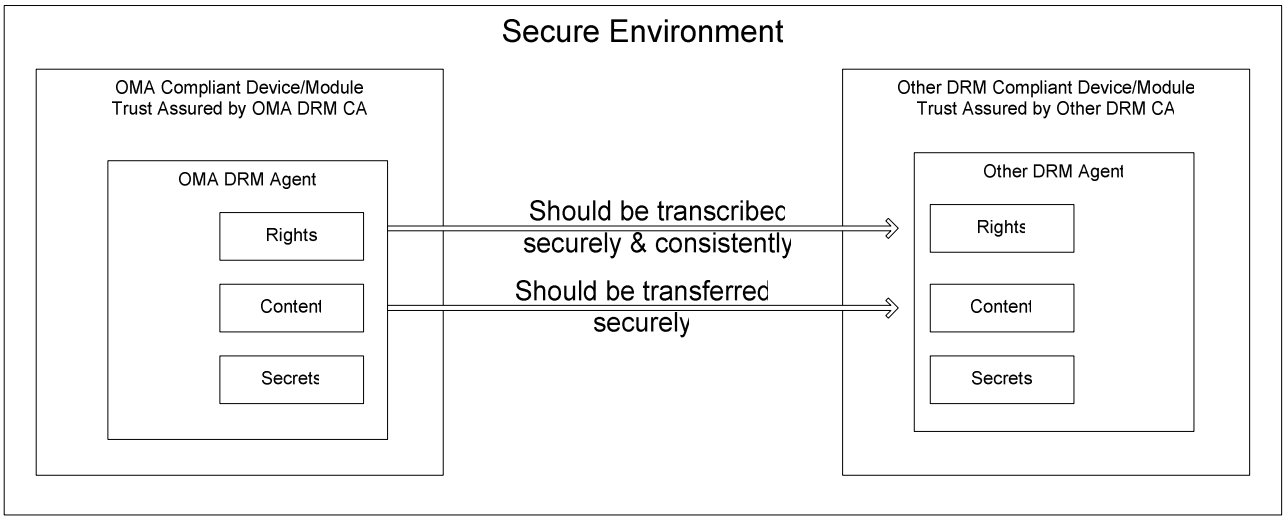

Figure 9: Exporting from OMA DRM

13.2 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 available for rendering 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 and 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 MUST NOT be transcribed into the other DRM system or content protection scheme. This restriction prevents further export once the content is protected by the other DRM system.

13.3 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 5.2.

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.

As described in section 9.2, a single Rights Object can contain rights for multiple content objects either within a multipart DCF or separate 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.

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.

13.4 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 unauthorized 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 signaling MUST be used to indicate to the target DRM/protection system, that the streamed content is used only for rendering purposes.

14.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 5.4.4.

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).

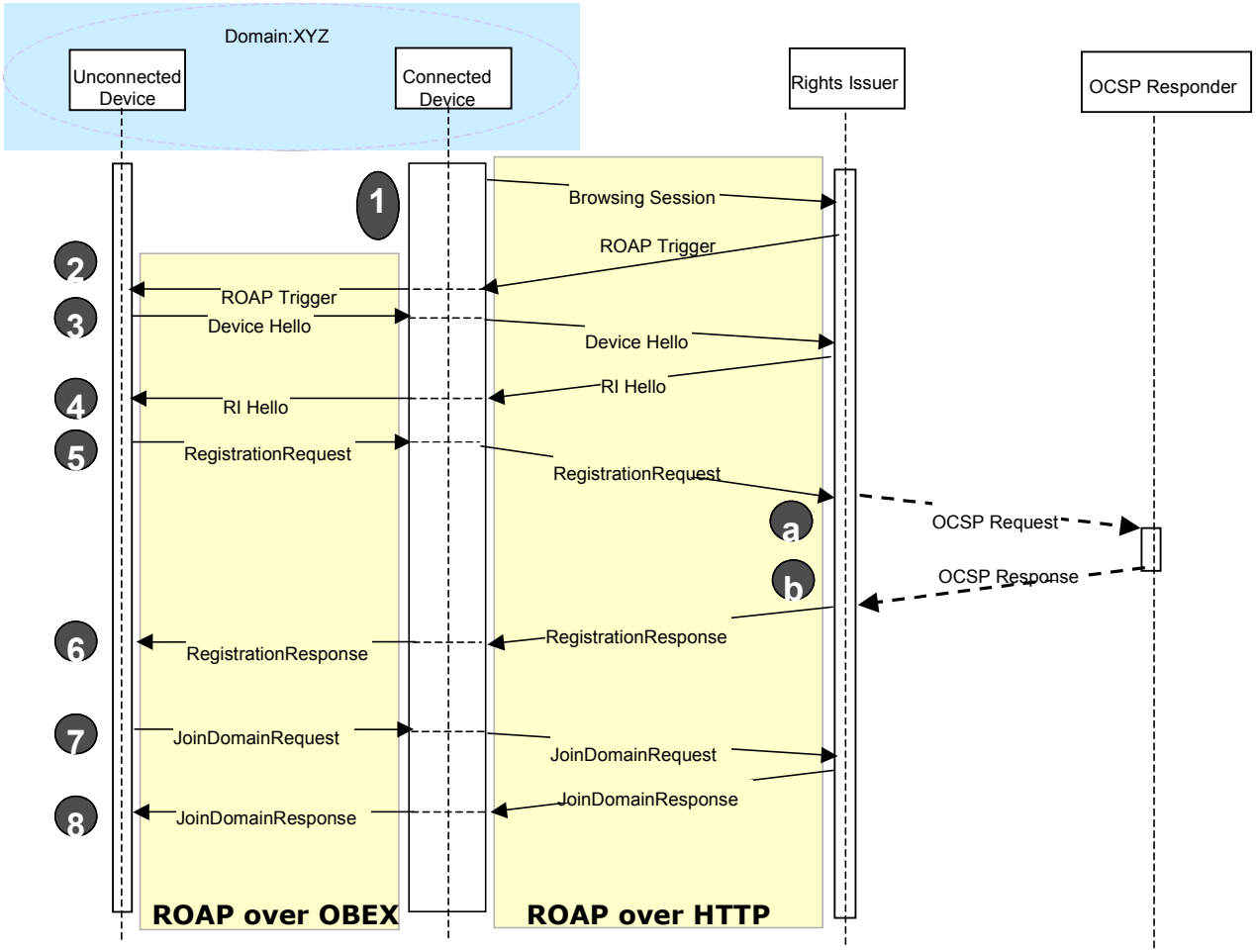

Figure 10: Unconnected Device Registration and Domain Establishment

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.

1. 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).
2. The RI returns a ROAP Trigger of type *joinDomain* to the Connected Device and includes the *proxy* attribute with the value set to "True".
3. 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.
4. 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.
5. 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
6. 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.
7. 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
8. 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.
9. 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.
10. 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.

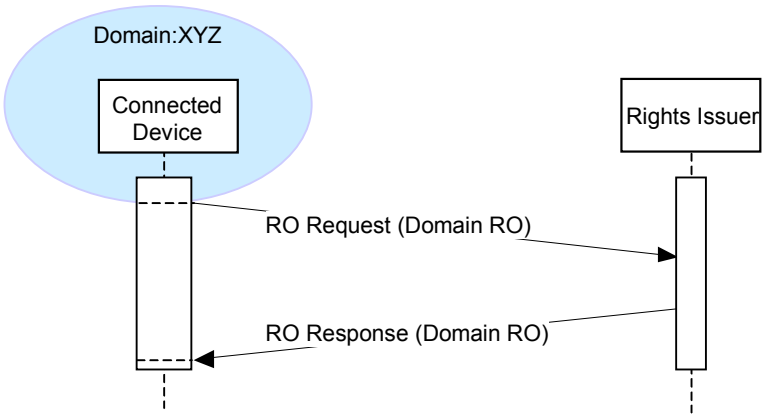

Figure 11: Content Acquisition

Once the Connected Device has received the Domain RO, it SHOULD insert the Domain RO in the associated DCF as specified in section 8.6.2.2. 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.

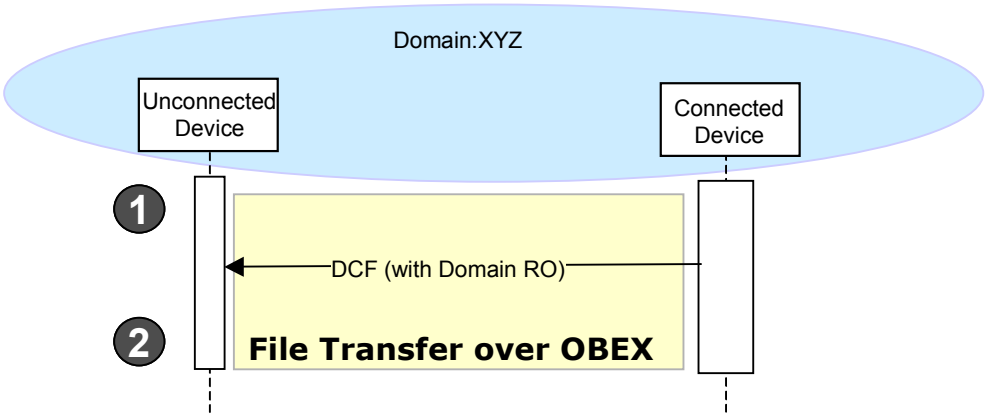

Figure 12: Content Acquisition

In the case where a user wishes to remove an Unconnected Device from a Domain, this is achieved as follows:

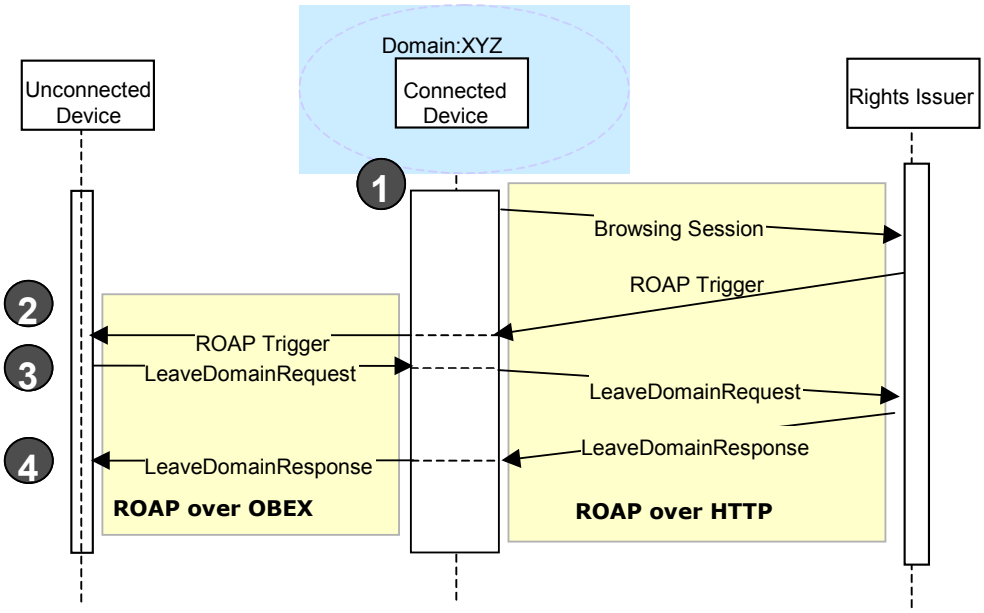

Figure 13: Unconnected Device leaving a Domain

1. 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".
2. 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.
3. 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.
4. 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.

15.Binding Rights to User Identities

15.1 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 [DRMRELv2], 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.

15.2 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.

1. The Device attempts to retrieve the user certificate from the WIM [WIM], identified by PKC_ID i.e. the value of PKC_ID matches with the value of CommonCertificateAttributes.certHash field from the user certificate CDF entry,
2. 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,
3. 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.

4. Generate a 20 bytes challenge value,
5. Ensure that the rights to access the WIM signature key are granted,
6. Request the WIM to sign the challenge using the private key associated with the identified user certificate,
7. 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.

15.2.1 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 D.

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 E.

16.Security Considerations (Informative)

16.1 Background

DRM solutions in general need to meet a number of security requirements. In particular, two requirements any DRM solution must fulfill are:

- Protected Content must only be accessed by properly authenticated and authorized DRM Agents
- Permissions on Protected Content must be honored by all DRM Agents

This specification along with its accompanying documents ([DRMARCH-v2], [DRMREL-v2], and [DRMCF-v2]) establishes the OMA DRM system. The OMA DRM system provides the means for the secure distribution and management of Protected Content in the OMA environment, and meets the requirements specified above and in [DRMREQ-v2].

16.2 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.

16.3 Security Mechanisms in the OMA DRM

16.3.1 Confidentiality

Confidentiality ensures that data is not accessible by an unauthorized party. As stated above, Protected Content must only be accessible by properly authenticated and authorized DRM Agents. To achieve this goal, Protected 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.

16.3.2 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, and the 2-pass Join Domain protocol. 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 Protected Content being wrapped with the recipient's public key, the initial requirement of Section 16.1 is still met. 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.

16.3.3 Integrity Protection

Data integrity protection ensures the ability to detect unauthorized modification of data. In the OMA DRM, data integrity protection, when applicable, is achieved through digital signatures on ROAP messages and Rights Objects.

16.3.4 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 unauthorized 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.

16.3.5 Other Characteristics

16.3.5.1 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 synchronized with the time held by an OCSP responder.

16.3.5.2 Transport Layer Security

The OMA DRM system provides application-layer security through the use of the security mechanisms listed in Section 16.3. Hence, it does not rely on, or assume, transport-layer security

16.3.5.3 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.

16.4 Threat Analysis

16.4.1 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 unauthorized behavior. In particular, since it may be in the interest of the user of the DRM agent to bypass the security, the DRM Agent must be robust against the "reversed" threat model. Besides protecting against the threat of a DRM entity compromise, the DRM system must protect against passive as well as active attacks.

In the following, it is assumed that an attacker is able to:

- Listen to the communication channel between a DRM Agent and a Rights Issuer, and
- Read, modify, remove, generate and inject messages in this channel.

When applicable, the case of a compromised DRM entity is also discussed.

16.4.2 Active Attacks

16.4.2.1 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 11.3 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).
- 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 11.3 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.
- Removal of a ROAP trigger before it reaches the Device will stop the intended ROAP protocol from being executed.

16.4.2.2 Message Modification

An attacker may modify any message sent between a DRM agent and an RI.

- For the Registration protocol, the RO Acquisition protocols, and the Join Domain 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.

16.4.2.3 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 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).
- ROAP triggers may be sent to any device at any time. Devices can protect against replay of **<leaveDomain>** triggers due to their digitally signed timestamp. For other triggers Devices cannot protect against replay attacks.
- 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 is achieved by means of the method specified in Section 9.4. It is important to note that the replay cache MUST be integrity-protected by the DRM Agent.

16.4.2.4 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].

16.4.2.5 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:
    i. The DRM agent's private key
    ii. Domain keys for any Domain the DRM Agent is a member of
    iii. Rights Object Encryption Keys
    iv. Content Encryption Keys
    v. Protected Content

  It may also result in loss of integrity protection of the DRM Agent's replay cache and/or loss of protection of Rights stored internally in the DRM Agent. Further it may result in loss of DRM Time, potentially allowing permissions to be overridden or compromised RIs to pose as uncompromised.

  Failure of DRM Agent implementations to protect the above assets may seriously compromise the security of the OMA DRM system and their protection is therefore critical.

  In addition, a compromised rendering application in the DRM Agent may also result in the loss of Protected Content. The DRM Agent implementation must therefore be robust and ensure that it only provides unprotected Protected Content to trusted rendering applications.

- A compromised Rights Issuer may result in the disclosure of any of the following:
    i. The Rights Issuer's private key
    ii. Domain keys for any Domain administered by the RI
    iii. Rights Object Encryption Keys
    iv. Content Encryption Keys
    v. Protected Content

  Again, the protection of these assets in RI implementations is crucial to the correct functioning of the OMA DRM.

- The effects on a PKI of a compromised CA or OCSP Responder is discussed, e.g., in [RFC3280] and [RFC2560].

The OMA DRM system relies on certificate revocation for minimizing the damages of a compromised entity. DRM Agents and RIs must always verify that the entity they are communicating with has not been compromised by checking the entity's certificate status. Further, in Domain settings, RIs may protect against undetected DRM agent compromise by regularly upgrading Domain Generations.

16.4.3 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 behavior 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 Protected Content, wrapped keys, or Rights Objects exchanged in such recorded messages.

16.5 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 behavior 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 Protected Content and associated RO, since the RI and CI are either administered by one organization 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 behavior, 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 behavior. In this case, however, the OMA DRM system does allow user policies to determine whether to use the Transaction ID mechanism or not. 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. ROAP Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<schema
 targetNamespace="urn:oma:bac:dldrm:roap-1.0"
 xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
 xmlns:o-ex="http://odrl.net/1.1/ODRL-EX"
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified">

<!-- Would import the OMA profile of ODRL if such a schema existed -->
<import namespace="http://odrl.net/1.1/ODRL-EX"/>

<import namespace="http://www.w3.org/2000/09/xmldsig#"
      schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-
schema.xsd"/>

<import namespace="http://www.w3.org/2001/04/xmlenc#"
      schemaLocation="http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/xenc-schema.xsd"/>

<!-- Basic Types -->
<complexType name="Request" abstract="true">
  <attribute name="triggerNonce" type="roap:Nonce"/>
</complexType>

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

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

<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"/>
  </restriction>
</simpleType>

<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>

<!-- ROAP extensions -->

<!-- No need for OCSPResponses to be sent -->
<!-- Mainly for use in the 2-pass RO Request protocol -->
<complexType name="NoOCSPResponse">
  <complexContent>
    <extension base="roap:Extension"/>
  </complexContent>
</complexType>

<!-- No need for receiving party's certificate chain to be sent -->
<!-- Mainly for use in the 2-pass RO Request protocol -->
<complexType name="PeerKeyIdentifier">
  <complexContent>
    <extension base="roap:Extension">
      <sequence minOccurs="0">
        <element name="identifier" type="roap:KeyIdentifier"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<!-- No need for inclusion of OCSP responder certificates -->
<!-- Mainly for use in the 2-pass RO Request protocol -->
<complexType name="OCSPResponderKeyIdentifier">
  <complexContent>
    <extension base="roap:Extension">
      <sequence>
        <element name="identifier" type="roap:KeyIdentifier"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<!-- Device Details -->
<!-- Used in ROAP-RIHello and ROAP-RegistrationRequest messages -->
<complexType name="DeviceDetails">
  <complexContent>
    <extension base="roap:Extension">
      <sequence minOccurs="0">
       <element name="manufacturer" type="string"/>
```

```
      <element name="model" type="string"/>
      <element name="version" type="string"/>
     </sequence>
    </extension>
  </complexContent>
</complexType>

<!-- Loyalty program information -->
<!-- Mainly for use in two-pass RO Request protocol-->
<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>

<!-- Certificate chain caching capabilities extension -->
<!-- (Device signals support of the extension, RI signals if it will -->
<!-- store the device's certificates) -->
<complexType name="CertificateCaching">
  <complexContent>
    <extension base="roap:Extension"/>
  </complexContent>
</complexType>

<!-- Support for hash chains in Domains -->
<complexType name="HashChainSupport">
  <complexContent>
    <extension base="roap:Extension"/>
  </complexContent>
</complexType>

<!-- Device does not consider itself a member of the domain it is -->
<!-- leaving -->
<complexType name="NotDomainMember">
  <complexContent>
    <extension base="roap:Extension"/>
  </complexContent>
</complexType>

<!—Domain Name Whitelist Extension for ROAP-RegistrationResponse -->
<!—RI can specify up to 5 domain names for silent/preview rights security -->
<complexType name="DomainNameWhiteList">
  <complexContent>
    <extension base="roap:Extension">
      <sequence maxOccurs="5">
       <element name="dn" type="string"/>
```

```
    </sequence>
   </extension>
  </complexContent>
</complexType>

<!-- Basic types to identify entities -->

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

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

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

<!-- Hash of complete DER-encoded subjectPublicKeyInfo from -->
<!-- key holder's 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 type -->
<element name="X509SPKIHash" type="roap:X509SPKIHash"/>

<!-- Domain Identifier -->
<!-- Last three characters (decimal digits) shall be interpreted as -->
<!-- domain generation -->
<simpleType name="DomainIdentifier">
  <restriction base="string">
    <pattern value=".{1,17}\d{3}"/>
  </restriction>
</simpleType>

<!-- 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"/>
```

```
  </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>

<!-- May be sent standalone (domain ROs) -->
<element name="protectedRO" type="roap:ProtectedRO"/>

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

<!-- Registration protocol -->

<!-- ROAP-DeviceHello -->
<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.
      association.
    </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>

<!-- 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" minOccurs="0"
            maxOccurs="unbounded"/>
      <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="hexBinary"/>
   </extension>
  </complexContent>
</complexType>

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

<!-- ROAP-RegistrationRequest -->
<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="hexBinary" use="required"/>
    </extension>
  </complexContent>
</complexType>

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

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

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

<!-- ROAP-RegistrationResponse -->
<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="hexBinary"/>
    </extension>
  </complexContent>
</complexType>

<!-- RO acquisition protocol -->

<!-- ROAP-RORequest -->
<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>

<!-- ROAP-ROResponse -->
<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>

<!-- Domain registration protocol -->

<!-- ROAP-JoinDomainRequest -->
<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="dateTime"/>
     <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>

<!-- ROAP-JoinDomainResponse -->
<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 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="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>

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

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

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

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

<!-- ROAP-LeaveDomainRequest -->
<element name="leaveDomainRequest" type="roap:DomainRequest"/>

<!-- ROAP-LeaveDomainResponse -->
<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 message.
    </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>

</schema>
```

Appendix B. 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 10.

Table 14: Backward Compatibility with Release 1.0

Appendix C. Application to Services (Informative)

C.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 standardization 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 15** explains the principle.

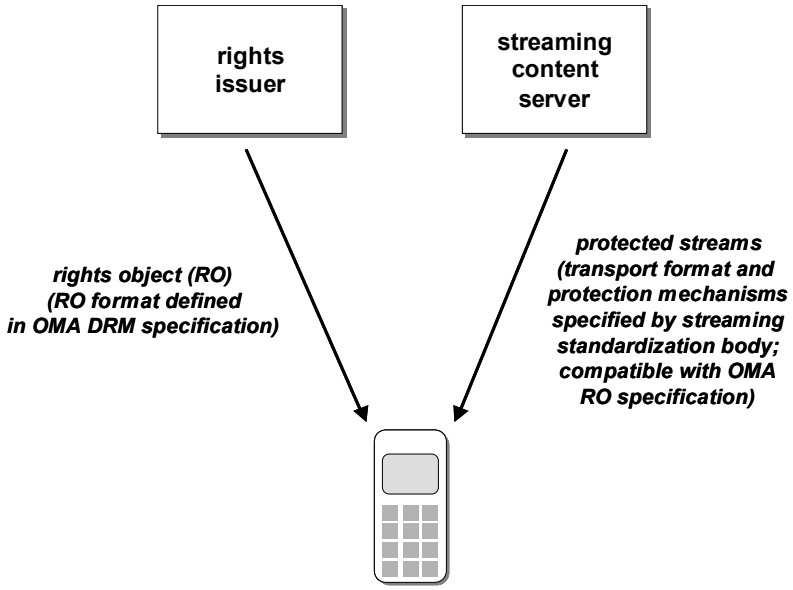

Figure 14: Generic principle of application of OMA DRM to streaming services

C.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 14, but there are some extensions that consider special features and properties of the PSS standard, namely

a) PSS sessions can consist of a mixture of discrete (e.g., JPEG images) and continuous (e.g., H.263 video) media

b) 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.

c) 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 packetized content as a special DCF, the Packetized DCF (PDCF) [DRMCF-v2]. It should thus be understood that a 3GP file holding encrypted tracks as defined in [TS26.244] is automatically a valid OMA DRM PDCF [DRMCF-v2].

Figure 15 gives an overview of the involved entities and data flows for DRM protection of 3GPP PSS sessions and media.

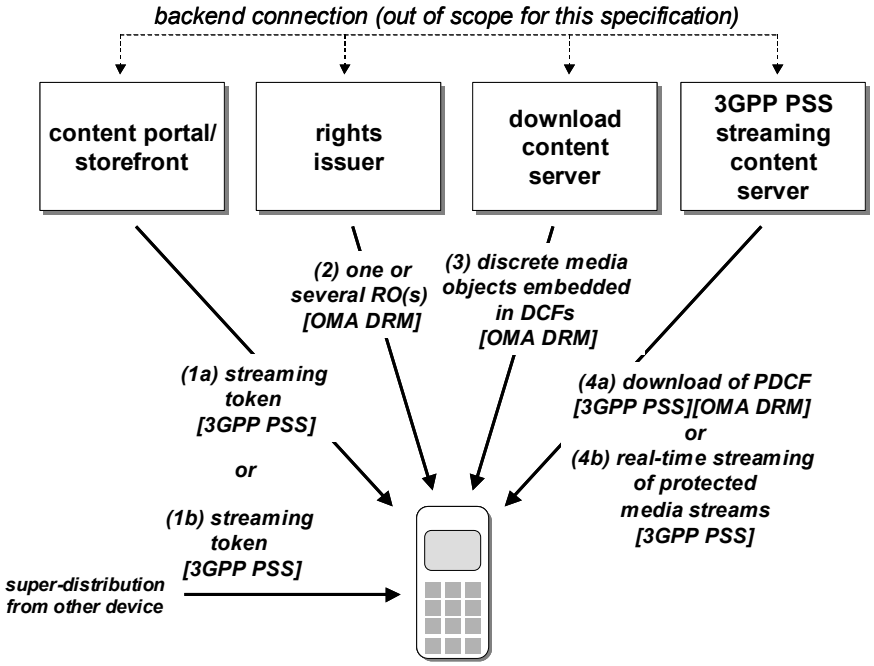

Figure 15: Application of OMA DRM to the 3GPP Packet-Switched Streaming Service (Release 6). References in brackets indicate where the respective data format or protocol is specified

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 15**).

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. 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 15), or by super-distribution (see (1b) in Figure 15), 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.

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 15). 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)

C. 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.

D. Depending on the type of streaming token, the following applies:

- SMIL presentation: Referenced discrete objects are downloaded from the respective download servers (see (3) and (4a) in Figure 15.). 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 15.). 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 optimization to fetch all ROs before starting the presentation.
- RTSP URL: Referenced streams are set up and started using PSS streaming protocols [3GPP PSS] (see (4b) in Figure 15). If ROs are not on the Device yet they can be acquired at this point, using the RI URL.
- SDP: Referenced streams are set up and started using PSS streaming protocols [3GPP PSS] (see (4b) in Figure 15). If ROs are not on the Device yet they can be acquired at this point, using the RI URL.

E. Discrete objects (DCFs), downloaded PSS content (PDCFs), and PSS streams are decrypted and rendered subject to the terms and permissions of the respective ROs.

F. 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).

C.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 16**.

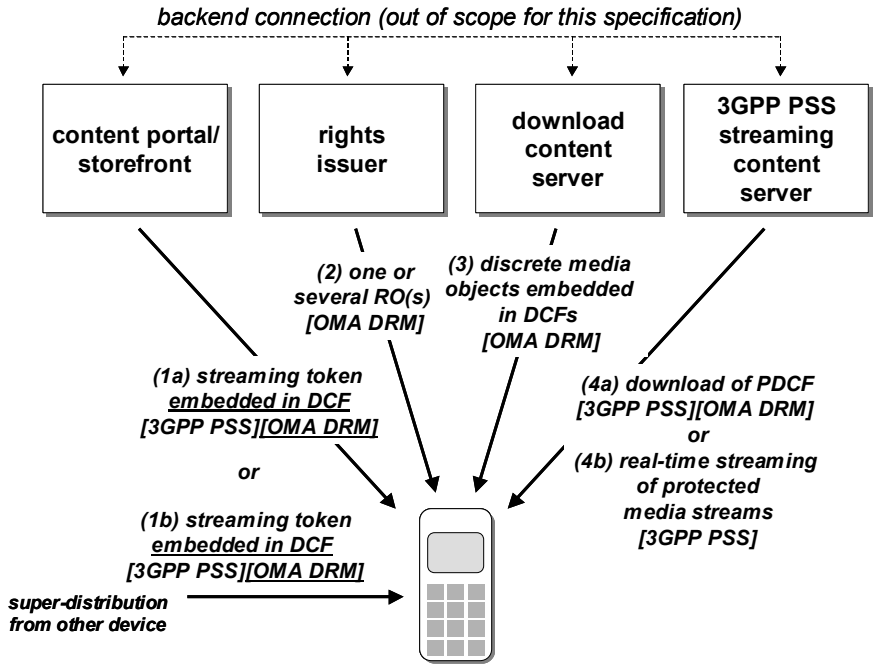

Figure 16: Application of OMA DRM to the 3GPP Packet-Switched Streaming Service (Release 6) with streaming token packaged into DCF. Underlined text denotes differences to Figure 15.

With this method, the typical steps to initiate, set up, receive, and render a protected 3GPP PSS session are similar as described in section C.1.1, with a few differences. The differences are outlined below.

(A) *Unchanged, see section C.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 16**). 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 C.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 16**). Referenced streams are set up and started using PSS streaming protocols [3GPP PSS] (see (4b) in **Figure 16**).

b. RTSP URL: Referenced streams are set up and started using PSS streaming protocols [3GPP PSS] (see (4b) in **Figure 16**). 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 16**).

(E) *Unchanged, see section C.1.1.*

(F) *Unchanged, see section C.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 recognized 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 D. Certificate Profiles and Requirements

D.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, organizationName, organizationalUnitName, commonName, and stateOrProvinceName.
Subject Name	MUST be present and MUST use a subset of the following naming attributes from [CertProf] – countryName, organizationName, organizationalUnitName, commonName, and serialNumber. The structure and contents of a Device subject name shall be as follows: [countryName=<Country of manufacturer>] [organizationName=<Manufacturer company name>] [organizationalUnitName=<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, organizationName, organizationalUnitName, 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), organizationName, organizationalUnitName, 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-organizations(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. 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; and
- MUST recognize and require the presence of the **`oma-kp-drmAgent`** object identifier defined above in the extKeyUsage extension in DRM Agent certificates.

D.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, organizationName, organizationalUnitName, commonName, and stateOrProvinceName.
Subject Name	MUST be present and MUST use a subset of the following naming attributes from [CertProf] – countryName, stateOrProvinceName, localityName, organizationName, organizationalUnitName, 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>] organizationName=<RI company name> [organizationalUnitName=<RI subsidiary/location>] commonName=<RI company name> "OMA Rights Issuer" [<serNo>] (For the commonName attribute, the <serNo> string is specified when a given organization has several RIs.) The countryName, organizationName, and commonName naming attributes must be present. The stateOrProvinceName, localityName, and/or organizationalUnitName 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, organizationName, organizationalUnitName, 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.

D.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.

D.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).

D.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 E. Interactions between the DRM Agent and the WIM(Informative)

E.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.

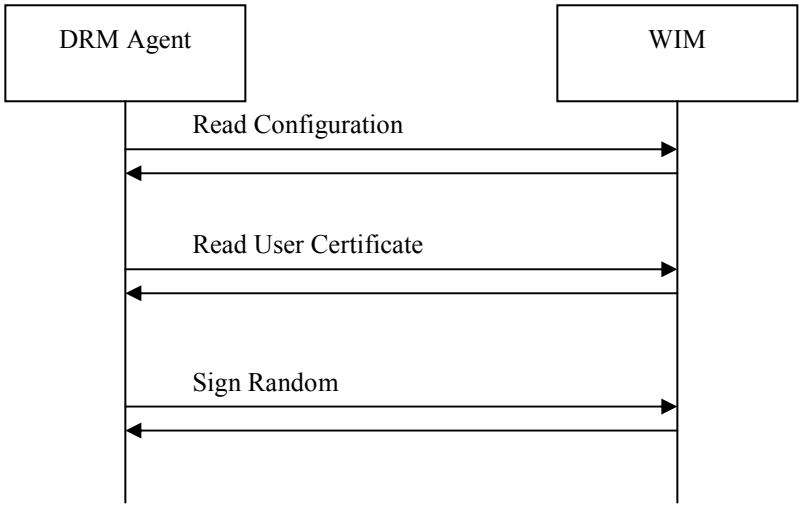

Figure 17: DRM Agent and WIM Interaction

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.

E.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 F. Static Conformance Requirements

The notation used in this appendix is specified in [IOPPROC].

F.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 V2.0 [DRMERELD-v2] defines the mandatory features to be supported by the Connected and Unconnected Devices. For further information, please see section 8 of [DRMERELD-v2].

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	5.3.10	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	5.3.8,5.3.9	M	
DRM-CLI-CMN-007	XML Canonicalization	5.3.9,5.4	M	Conforming to [XC14N]
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, 5.4.4	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	5.1.4,5.1.5, 5.4.4,7.2.3, 7.3,8	O	DRM-CLI-CMN-016,DRM-CLI-CMN-031,DRM-CLI-CMN-032,DRM-CLI-CMN-033,DRM-

				CLI-CMN-034,DRM-CLI-CMN-035,DRM-CLI-CMN-042
DRM-CLI-CMN-016	Hash Chains for Domain Key Management	5.4.4.1.1,7.3,8.7.1	O	
DRM-CLI-CMN-017	DRM Agent Certificates	D.1	M	
DRM-CLI-CMN-018	User Certificates for WIM Binding	D.5	O	
DRM-CLI-CMN-019	RI Certificate Processing and Certificate Chain Validation	5.4.2.4,5.4.3.2,5.4.4.2,6.2	M	
DRM-CLI-CMN-020	RI Signature Validation	5.4.2.4,5.4.3.2,5.4.4.2	M	
DRM-CLI-CMN-021	OCSP Response Validation	5.4.2.4,5.4.3.2,5.4.4.2,6.2,6.3	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	15.1	O	
DRM-CLI-CMN-023	WIM Binding	15.2	O	DRM-CLI-CMN-018
DRM-CLI-CMN-024	Transaction Tracking	12.3, 5.4.3.1, 5.4.3.2.1	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	9.4,5.3.9	M	
DRM-CLI-CMN-030	Maintaining state information for Stateful Rights Objects	9.4.1	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	5.4.4.2.1,8.2	O	
DRM-CLI-CMN-034	Domain Context Expiry processing	5.4.4.2.1	O	
DRM-CLI-CMN-035	Installing Domain ROs	8.6.2.1, 8.6,5.4.4.2	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	5.4.2.4.1,5.3.8,5.2.1	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	5.4.3.1.1	O	
DRM-CLI-CMN-041	Device RO Processing	9.3.1	M	
DRM-CLI-CMN-042	Domain RO Processing	8.6	O	
DRM-CLI-CMN-043	MIME Types for ROAP PDU, Trigger, ProtectedRO, and Rights Objects	5.3.8,10.2	M	
DRM-CLI-CMN-044	Exporting to other DRMs and Protected Links	13	O	
DRM-CLI-CMN-045	Super Distribution of the DCF	12	O	
DRM-CLI-CMN-046	Super Distribution of the ContentURL	12	O	
DRM-CLI-CMN-047	Parent Rights Object	9.5	M	
DRM-CLI-CMN-048	Off-device storage of content and Rights Objects	9.6	O	
DRM-CLI-CMN-049	Capability signaling to Content Issuers and Rights Issuers	10	M	
DRM-CLI-CMN-050	Processing Content Objects, Rights Objects and ROAP Triggers received via WAP PUSH	11.4	M	
DRM-CLI-CMN-051	DCF Integrity protection after the DCFs are downloaded to the Device	12.4	M	
DRM-CLI-CMN-052	Backwards Compatibility to OMA DRM v1	Appendix B	M	

DRM-CLI-CD-053	DRM Time	6.3,5.4	O	DRM-CLI-CD-054
DRM-CLI-CD-054	DRM Time Synchronization	6.3,5.4	O	
DRM-CLI-CD-055	Connectivity for Unconnected Devices via ROAP over OBEX	11.6	O	
DRM-CLI-CD-056	Connectivity to Rights Issuers over appropriate transport connections	14	O	
DRM-CLI-CD-057	2-pass ROAP-ROAcquisition protocol	5.4.3	O	
DRM-CLI-CD-058	1-pass ROAP-ROAcquisition protocol	5.4.3.2.1	O	
DRM-CLI-CD-059	2-pass ROAP-JoinDomain protocol	5.4.4.1	O	
DRM-CLI-CD-060	2-pass ROAP-LeaveDomain protocol	5.4.4.3	O	
DRM-CLI-CD-061	HTTP Transport Mapping	11.2	O	
DRM-CLI-CD-062	Capability Signalling	10	O	
DRM-CLI-CD-063	Silent and Preview header processing in DCFs	5.2.2	O	
DRM-CLI-CD-064	Download OTA support for delivering Content , ROAP Triggers, and Rights Objects	11.3	O	
DRM-CLI-UD-065	Utilize the connectivity provided by the Connected Device to conduct ROAP protocols	14	O	
DRM-CLI-UD-066	ROAP-OBEX Server	14,11.6	O	
DRM-CLI-UD-067	2-pass ROAP JoinDomain protocol	5.4.4.1	O	
DRM-CLI-UD-068	2-pass ROAP LeaveDomain protocol	5.4.4.3	O	

F.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	5.3.10	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 Canonicalization	5.3.9,5.4	M	
DRM-SERVER-006	RI Certificates	D.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	domainID 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	5.4.3.2.1	M	
DRM-SERVER-018	2-pass ROAP-JoinDomain Protocol	5.4.4.1	M	
DRM-SERVER-019	2-pass ROAP-LeaveDomain Protocol	5.4.4.3	M	
DRM-SERVER-020	Hash Chain support for Domain Key Generation	8.7.1	O	
DRM-SERVER-021	ProtectedRO support	5.3.8	M	

[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.

DRM-SERVER-022	Signature on Domain RO	5.4.3.2.1,5.3.9	M	
DRM-SERVER-023	Signature on Device RO	5.3.9,5.4.3.2.1	O	
DRM-SERVER-024	domainRO and riURL attributes in ProtectedRO for Domain ROs	5.3.9	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	5.3.9	M	
DRM-SERVER-031	Parent Rights Object	9.5	M	
DRM-SERVER-032	Issuer Responsibilities	10.4	M	
DRM-SERVER-033	Download OTA support for delivering Content , ROAP Triggers, and Rights Objects	11.3	O	
DRM-SERVER-034	Use of WAP PUSH to deliver Content, ROAP Triggers, and Rights Objects	11.4	M	
DRM-SERVER-035	Transaction Tracking	12.3, 5.4.3.1, 5.4.3.2.1	M	

Appendix G. Examples (Informative)

G.1 ROAP Examples

All examples are syntactically correct. Signature, MAC, cipher and digest values are fictitious however.

G.1.1 Device hello

```
<?xml version="1.0" encoding="utf-8"?>
<roap:deviceHello
 xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <version>1.0</version>
  <deviceID>
    <keyIdentifier xsi:type="roap:X509SPKIHash">
      <hash>vXENc+Um/9/NvmYKiHDLaErK0gk=</hash>
    </keyIdentifier>
  </deviceID>
  <extensions>
    <extension xsi:type="roap:CertificateCaching"/>
  </extensions>
</roap:deviceHello>
```

G.1.2 RI Hello

```
<?xml version="1.0" encoding="utf-8"?>
<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>1.0</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>
```

G.1.3 Registration Request

```
<?xml version="1.0" encoding="utf-8"?>
<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>
```

G.1.4 Registration Response

```
<?xml version="1.0" encoding="utf-8"?>
<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="www.anyRI.com/register">
   <riURL>http://ri.example.com/roap.cgi</riURL>
   <certificateChain>
      <certificate>miib123121234567</certificate>
      <certificate>miib234124312431</certificate>
   </certificateChain>
   <ocspResponse>fdow9rw0feijfdsojr3w09u3wijfdslkj4sd</ocspResponse>
   <signature>321ue3ue3ue10ue2109ue1ueoidwoijdwe309u09ueqijdwqijdwq09uwqwqi009</signature>
</roap:registrationResponse>
```

G.1.5 RO Request

The request is for a Device RO.

```
<?xml version="1.0" encoding="utf-8"?>
<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>
```

G.1.6 RO Response

The response is a Rights Object intended for the recipient only. Note that the response indicates that the Rights Object is stateful.

```
<?xml version="1.0" encoding="utf-8"?>
<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>
  <protectedRO>
    <ro id="n8yu98hy0e2109eu09ewf09u" stateful="true" version="1.0">
      <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>2.0</o-dd:version>
          <o-dd:uid>RightsObjectID</o-dd:uid>
        </o-ex:context>
        <o-ex:agreement>
          <o-ex:asset>
            <o-ex:context>
              <o-dd:uid>ContentID</o-dd:uid>
            </o-ex:context>
            <o-ex:digest>
              <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
              <ds:DigestValue>bLLLc+Um/5/NvmYKiHDLaErK0fk=</ds:DigestValue>
            </o-ex:digest>
            <ds:KeyInfo>
              <xenc:EncryptedKey>
                <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
                <xenc:CipherData>
                  <xenc:CipherValue>EncryptedCEK</xenc:CipherValue>
                </xenc:CipherData>
              </xenc:EncryptedKey>
              <ds:RetrievalMethod URI="#K_MAC_and_K_REK"/>
            </ds:KeyInfo>
```

```
            </o-ex:asset>
            <o-ex:permission>
              <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>
      </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:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:SignatureValue>
        <ds:KeyInfo>
          <ds:RetrievalMethod URI="#K_MAC_and_K_REK"/>
        </ds:KeyInfo>
      </mac>
    </protectedRO>
    <ocspResponse>miibewqoidpoidsa</ocspResponse>
    <extensions>
      <extension xsi:type="roap:TransactionIdentifier">
        <id>09321093209-2121</id>
      </extension>
    </extensions>
    <signature>d93e5fue3susdskjhkjedkjrewh53209efoihfdse10ue2109ue1</signature>
</roap:roResponse>
```

G.1.7 Domain RO

The Domain RO may be sent separately, as here, or within a ROAP-ROResponse.

```
<?xml version="1.0" encoding="UTF-8"?>
<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">
<ro id="n8yu98hy0e2109eu09ewf09u" domainRO="true" version="1.0"
   riURL="http://www.ROs-r-us.com">
  <rilD>
    <keyIdentifier xsi:type="roap:X509SPKIHash">
      <hash>aXENc+Um/9/NvmYKiHDLaErK0fk=</hash>
    </keyIdentifier>
  </rilD>
  <rights o-ex:id="REL1">
    <o-ex:context>
      <o-dd:version>2.0</o-dd:version>
      <o-dd:uid>RightsObjectID</o-dd:uid>
    </o-ex:context>
    <o-ex:agreement>
      <o-ex:asset>
        <o-ex:context>
         <o-dd:uid>ContentID</o-dd:uid>
        </o-ex:context>
        <o-ex:digest>
         <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
         <ds:DigestValue>bLLLc+Um/5/NvmYKiHDLaErK0fk=</ds:DigestValue>
        </o-ex:digest>
        <ds:KeyInfo>
         <xenc:EncryptedKey>
          <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
          <xenc:CipherData>
           <xenc:CipherValue>EncryptedCEK</xenc:CipherValue>
          </xenc:CipherData>
         </xenc:EncryptedKey>
         <ds:RetrievalMethod URI="#K_MAC_and_K_REK"/>
        </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:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</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:DomainIdentifier>Domain-XYZ-001</roap:DomainIdentifier>
    </ds:KeyInfo>
    <xenc:CipherData>
     <xenc:CipherValue>32fdsorew9ufdsoi09ufdskrew9urew0uderty5346wq</xenc:CipherValue>
    </xenc:CipherData>
   </encKey>
  </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:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
     <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</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>
```

G.1.8 Join Domain Request

```
<?xml version="1.0" encoding="utf-8"?>
<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>
```

G.1.9 Join Domain Response

```
<?xml version="1.0" encoding="utf-8"?>
<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>
      <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>
      </domainKey>
    </domainInfo>
    <certificateChain>
      <certificate>MIIB223121234567</certificate>
      <certificate>MIIB834124312431</certificate>
    </certificateChain>
    <ocspResponse>miibewqoidpoidsa</ocspResponse>
    <signature>d93e5fue3ue10ue2109ue1ueoidwoijdwe309u09ueqijdwqijdwq09uwqwqi009</signature>
</roap:joinDomainResponse>
```

G.1.10 Leave Domain Request

```
<?xml version="1.0" encoding="utf-8"?>
<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>
```

G.1.11 Leave Domain Response

```
<?xml version="1.0" encoding="utf-8"?>
<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>
```

G.1.12 Roap Trigger

This example is for a "Leave Domain" trigger.

```
<?xml version="1.0" encoding="UTF-8"?>

<roap-trigger:roapTrigger
  xmlns:roap-trigger="urn:oma:bac:dldrm:roap-trigger-1.0"
  xmlns:roap="urn:oma:bac:dldrm:roap-1.0"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.0">
  <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:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

```
      <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</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:DomainIdentifier>Domain-XYZ-001</roap:DomainIdentifier>
  </ds:KeyInfo>
  <xenc:CipherData>
    <xenc:CipherValue>32fdsorew9ufdsoi09ufdskrew9urew0uderty5346wq</xenc:CipherValue>
  </xenc:CipherData>
 </encKey>
</roap-trigger:roapTrigger>
```

G.2 HTTP Transport Mapping Examples

G.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).

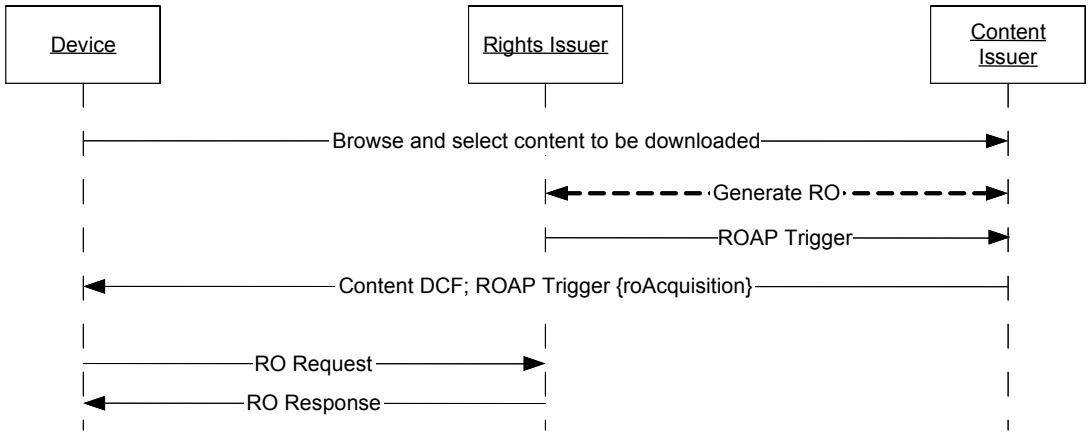

Figure 18: Separate Delivery of DCF and RO

- 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 acqusition of the Right Object and delivers the ROAP Trigger to CI.
- The CI returns an HTTP Response containing a multipart/mixed. 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/mixed; boundary=”XX---XX”

--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
 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] ...
```

G.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.

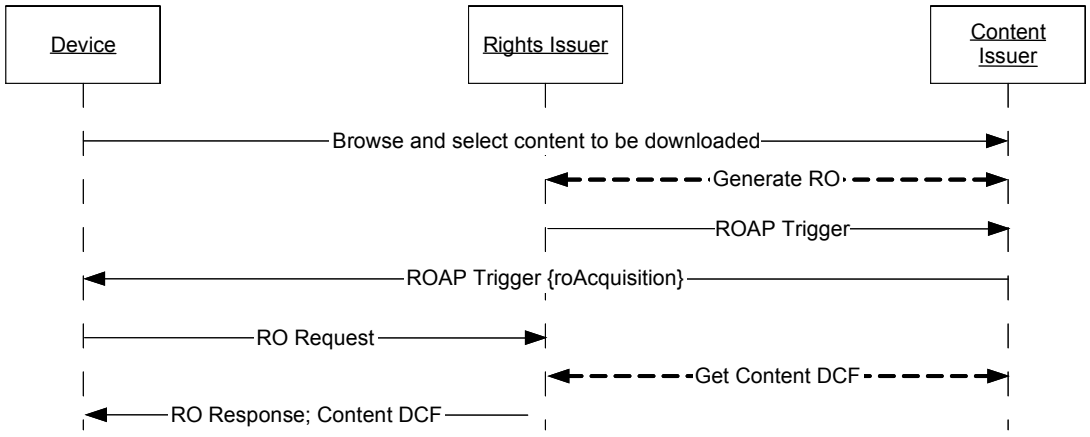

Figure 19: Combined Delivery of DCF and RO

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 acqusition 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
 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"

--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—
```

G.2.3 Silent RO Acquisition Triggered by DCF Headers

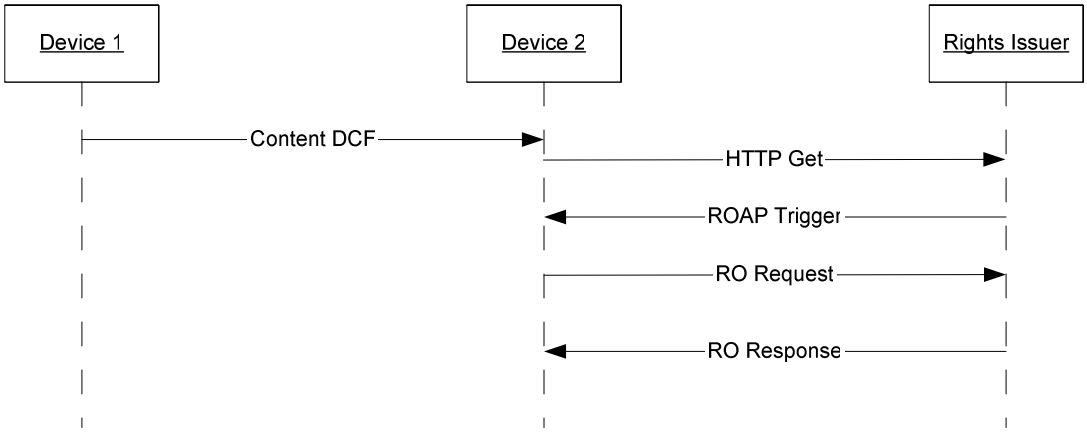

Figure 20: Silent RO Acquisition Triggered by DCF Headers

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
 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] ...
```

G.3 Download OTA Examples

G.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.

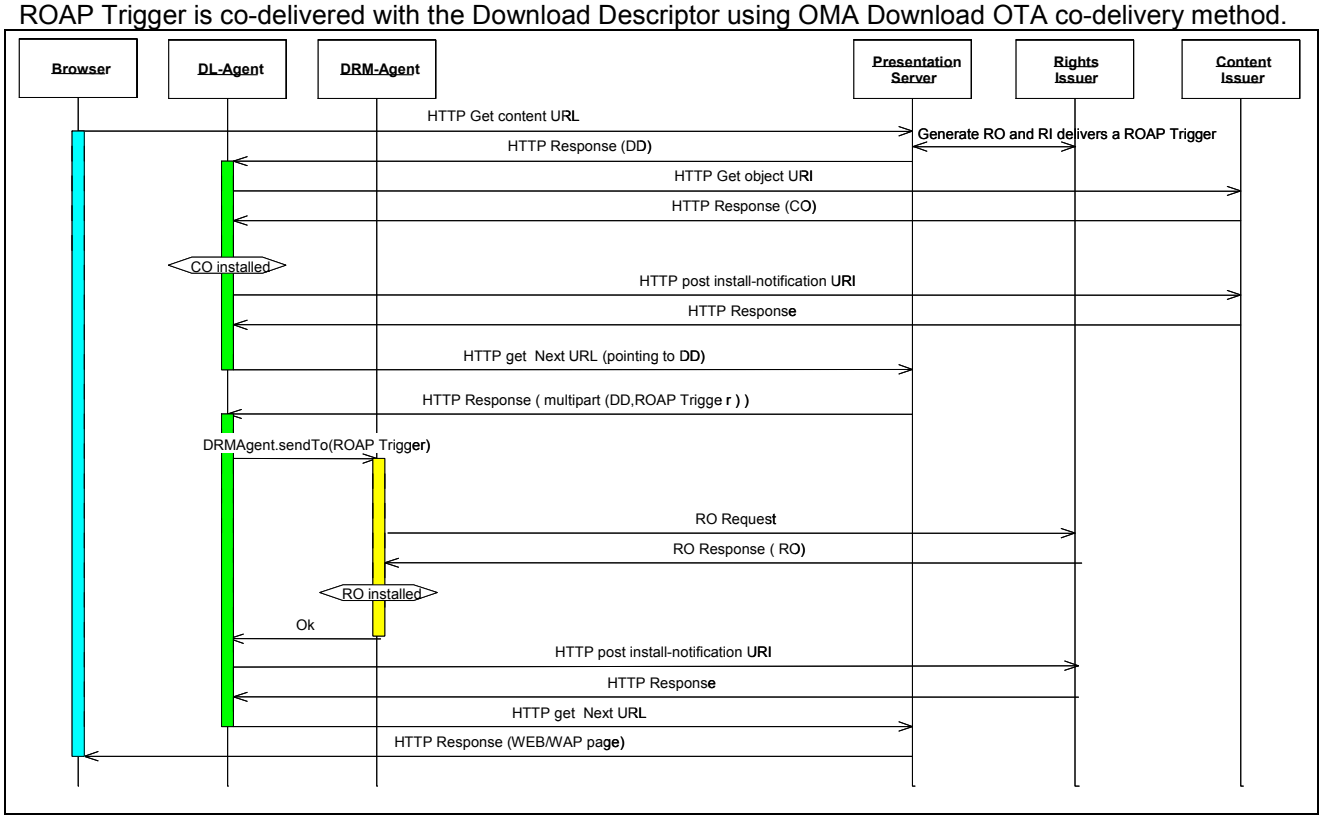

Figure 21: Using Download OTA to deliver DRM Content and Rights Object

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 acqusition 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"
 --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 xmlns="urn:oma:bac:dldrm:roap-trigger-1.0">
  <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,
application/vnd.oma.drm.ro+xml
 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
```

G.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.

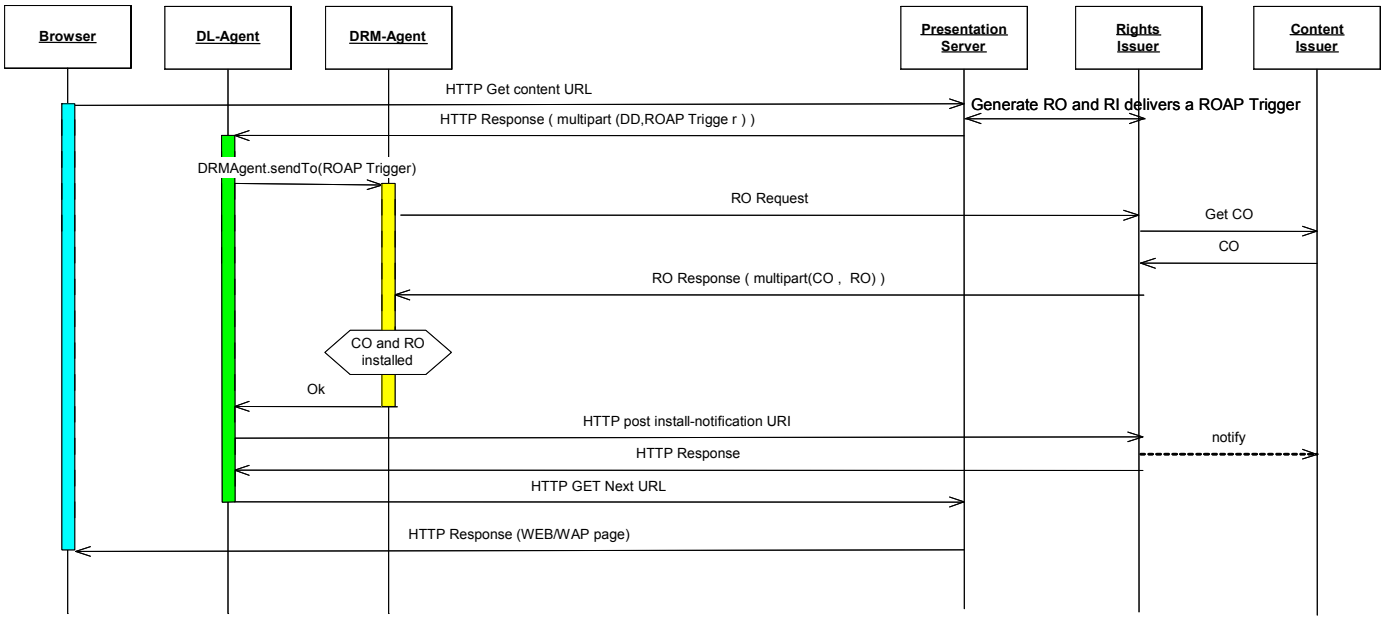

Figure 22: Combined Delivery of DRM Content and Rights Object

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 acqusition 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"
 --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:sd89632341@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: <sd89632341@ri.example.com>
```

```
Content-Type: application/vnd.oma.drm.roap-trigger+xml
<roapTrigger xmlns="urn:oma:bac:dldrm:roap-trigger-1.0">
   <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,
application/vnd.oma.drm.ro+xml, application/vnd.oma.drm.dcf
 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"

--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
```

G.4 MMS Examples

G.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

--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--
```

G.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.

G.5.1 ROAP Trigger

This message is sent from the Connected Device to the Unconnected Device after:

1. The Connected Device has received the trigger from the RI;
2. The Connected Device has determined that the ROAP Trigger is not for itself; and
3. 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

Bytes	Meaning
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…

G.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

G.6 Example – Exporting to Removable Media

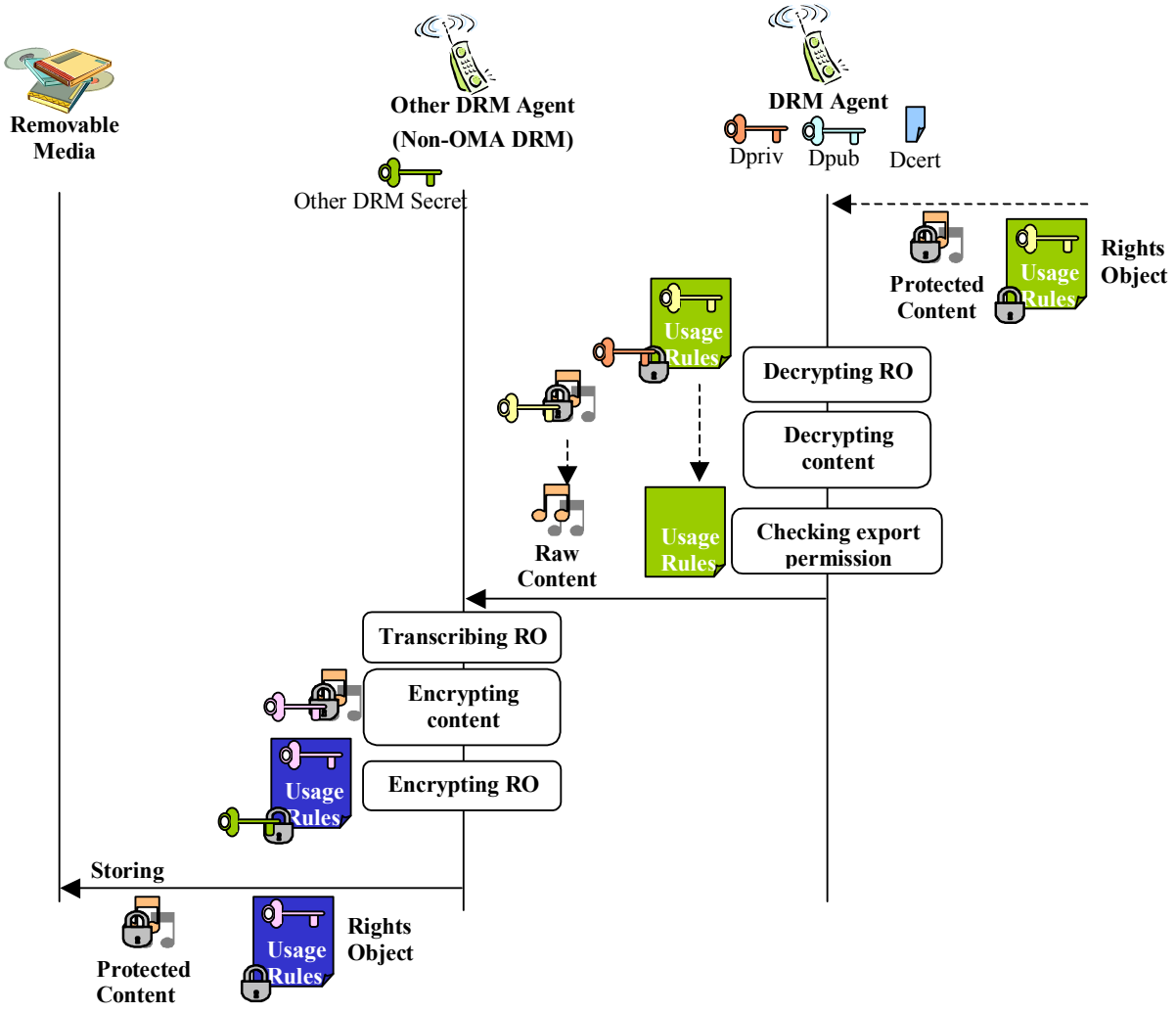

Figure 23:. Example – Exporting to Removable Media

An example of exporting DRM protected content and Rights Object to other DRM (non-OMA DRM) system, which has some authorized protection mechanism, is shown above.

1. After Protected Content and protected Rights Object are delivered to a trusted OMA DRM Agent, the Protected 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 protected content with CEK that is derived from decrypted Rights Object.
2. 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 Protected Content and RO are suitable for the targeted DRM capability.
3. The raw content and usage rules are transferred from OMA DRM Agent to the other DRM Agent.

4. 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.

5. 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.
6. 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.
7. 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.

Appendix H. Change History (Informative)

H.1 Approved Version History

Reference	Date	Description
n/a	n/a	No prior version

H.2 Draft/Candidate Version 2.0 History

Document Identifier	Date	Sections	Description
Candidate Versions OMA-DRM-DRM-V2_0	16 Jul 2004	n/a	Candidate version.
	10 Dec 2004	Various, according to the associated CRs	Following CRs were incorporated for this update to the Candidate Spec. CR 222 : Join Domain Response CR 206 : Leave Domain clarification CR 189 : Normative references to RSA-PSS CR 200 : Algorithm URL correction CR 190 : Multiple Rights for Multipart DCFs CR 192R02 : Domain Generation in Definitions CR 193R04 : Encoding ambiguity in KDF CR 213 : Domain Name Whitelist Schema Snippet in 5.4 CR 216R01 : Certificate Chain clarification in 5.4 CR 217R01 : PSS RSA-PSS Definitions fix CR 220 : Retrieval method correction CR 225 : Clarification of mathematical notations CR 175R01 : ProtectedDomainKey schema fix CR 172R01 : hash algorithm for DCF hash CR 231 : ROAP PDU XML clarification CR 252: riURL DeviceTimeError CR 240R01 : Algorithm Definitions clarifications CR 265R01 : Free Space and hash calculation related fix CR 233R01 : OCSP MP conformance SCR changes CR 268R01 : ProtectedRO example angle bracket fix CR 246R01 : Optional content hash in progressive download scenario CR 264R01 : Transaction tracking and group ID clarifications CR 198R02 : Group ID clarifications, add section 9.7 CR 205 : Transaction Tracking Clarification CR 165 : Silent Retrieval CR 214R04 : ContentID in ROAP trigger CR 266R02 : ROAP error message to unknown or malformed requests CR 174R02 : ROAP-Trigger and examples TP ref #OMA-TP-2005-0054-INP_Notifcation_of-CRs_to_DRM2