OMA Secure Removable Media Specification

Approved Version 1.1 – 16 Aug 2011

Open Mobile Alliance

OMA-TS-SRM-V1_1-20110816-A

Contents

Figures

Tables

1. Scope

The scope of OMA “Secure Removable Media” is to enable the use of the Secure Removable Media based on the OMA DRM version 2.0, DRM version 2.1, OMA DRM XBS [DRMXBSv1.1] and OMA SCE 1.0 (especially the SCE DRM and SCE REL) specifications. This specification defines mechanisms and protocols necessary to implement the Secure Removable Media and the extended parts of the OMA DRM and the OMA SCE system to enable the use of the Secure Removable Media.

2. References

2.1 Normative References

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

[BCAST1.1-SPCP] "Service and Content Protection for Mobile Broadcast Services", version 1.1, Open Mobile Alliance™, OMA-BCAST-SvcCntProtection-V1_1.
URL: http://www.openmobilealliance.org/

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

[DRMXBSv1.1] "OMA DRM v2.0 Extensions for Broadcast Support". Version 1.1. Open Mobile Alliance™. OMA-TS-DRM_XBS-V1_1.
URL: http://www.openmobilealliance.org/

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

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

[ISO8601] "Data elements and interchange formats -- Information interchange -- Representation of dates and times". ISO 8601:2004.
URL: http://www.iso.org

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

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

[OMADRMv2.0] "Digital Rights Management". Version 2.0. Open Mobile Alliance™. OMA-DRM-DRM-V2_0.
URL: http://www.openmobilealliance.org/

[OMADRMv2.1] "Digital Rights Management". Version 2.1. Open Mobile Alliance™. OMA-DRM-DRM-V2_1.
URL: http://www.openmobilealliance.org/

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

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

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

[RFC2396] "RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax". T. Berners-Lee, R. Fielding, L. Masinter. August 1998.
URL: http://www.ietf.org/rfc/rfc2396.txt

[RFC2630] "RFC 2630: Cryptographic Message Syntax". R. Housley. June 1999.
URL: http://www.ietf.org/rfc/rfc2630.txt

[RFC3280] "RFC 3280: Internet Public Key Infrastructure - Certificate and Certificate Revocation List (CRL) Profile". R. Housley, W. Polk, W. Ford, and D. Solo. April 2002.
URL: http://www.ietf.org/rfc/rfc3280.txt

[SCE-DRM] "DRM Specification – SCE Extensions, Draft Version". Open Mobile Alliance™. OMA-TS-SCE_DRM-V1_0-D.
URL: http://www.openmobilealliance.org/

[SCE-REL] "DRM Rights Expression Language – SCE Extensions, Draft Version". Open Mobile Alliance™. OMA-TS-SCE_REL-V1_0-D.
URL: http://www.openmobilealliance.org/

[SHA1] "NIST FIPS 180-2: Secure Hash Standard". August 2002.
URL: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

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

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

2.2 Informative References

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

[ISO/IEC13818-1] "ISO/IEC 13818-1, Information technology - Generic coding of moving pictures and associated audio information - Part 1: Systems". December 2000.
URL: http://www.iso.org

[SRM-ADv1.1] "OMA Secure Removable Media Architecture". Open Mobile Alliance™. OMA-AD-SRM-V1_1.
URL: http://www.openmobilealliance.org/

[SRM-RDv1.1] "OMA Secure Removable Media Requirements". Open Mobile Alliance™. OMA-RD-SRM-V1_1.
URL: http://www.openmobilealliance.org/

3. Terminology and Conventions

3.1 Conventions

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

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

3.2 Definitions

BCAST Rights	BCAST Rights are the collection of permissions and constraints governing access to DRM Content in Mobile Broadcast Services (see DRM profile of the BCAST Service and Content Protection Specification [BCAST1.1-SPCP]). BCAST Rights are identical to Rights in the context of this specification. Encoding of the RO is specified in [OMADRMv2.0], and some extensions are specified in [DRMXBSv1.1]
BCAST Token	A virtual currency used for token based consumption of BCAST content. It is the same as the token from [DRMXBSv1.1].
BCAST Token Container	A collection of BCAST Tokens that can be transferred from one Device to another.
Broadcast Rights	Broadcast Rights govern access to DRM Content and are delivered to the Device using the broadcast channel (see [DRMXBSv1.1]). For the purpose of this specification, Broadcast Rights consist of a Broadcast RO, its associated state, and other related information.
Broadcast Rights Object	This is a Rights Object used by the DRM profile of the Service and Content Protection specification [OMA-BCAST-SCP]. BCRO is delivered over a broadcast channel. Encoding of the BCRO is specified in Section 8 of the XBS specification [DRMXBSv1.1].
Broadcast RO Move	To Move the Broadcast RO between SRM and Device.
Composite Object	A content object that contains one or more Media Objects by means of inclusion. (From [OMADRMv2.0])
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. The Device may include a smartcard module (e.g. a SIM) or not depending upon implementation.
DRM Agent	The entity in the Device that manages Permissions for Media Objects on the Device. (From [OMADRMv2.0])
DRM Content	Media Objects that are consumed according to a set of Permissions in a Rights Object. (From [OMADRMv2.1])
Local Rights Consumption	Operations in which Rights stored in SRMs are transferred for use by the recipient Device for a limited period of time for rendering purposes.
Media Object	A digital work e.g. a ringing tone, a screen saver, a Java game or a Composite Object (From [OMADRMv2.0])
Move	To make Rights existing initially on a source Device or SRM fully or partially available for use by a recipient Device or SRM, such that the Rights or parts thereof that become usable on the recipient Device or SRM can no longer be used on the source Device or SRM.
Move Count	Counter for the number of Moves that still can be performed. The Move Count only exists for a Move permission/action with a count constraint, since for other permissions/actions there is no need to maintain a Move Count.
Generalised Rights Object	This term is used in this document as a more generic term whenever an RO or a BCRO is meant.
Handle	A random number generated by the DRM Agent, which is stored in the SRM and in the Operation Log (kept in the Device) used for associating the DRM Agent to specific Rights for the Move or Local Rights Consumption operation.
Operation Log	A secure file, kept in a Device, in which entries containing transaction information (e.g. ROID, Handle) are stored until corresponding transactions are completed. The information in an entry is relevant for the recovery

	procedures used by a DRM Agent when a transaction is not completed.
Permission	Actual usages or activities allowed (by the Rights Issuer) over DRM Content. (From [OMADRMv2.1])
Provisioning	To provision Rights directly to an SRM.
Rights	Rights are the collection of permissions and constraints defining under which circumstances access is granted to DRM Content. For the purposes of this document, Rights consist of a Rights Object, its associated state, and other related information.
Rights Issuer	An entity that issues Rights Objects to OMA DRM conformant Devices. (From [OMADRMv2.1])
Rights Move between SRM and Device	To Move the Rights between SRM and Device.
Rights Object	A collection of Permissions and other attributes which are linked to DRM Content. (From [OMADRMv2.1]) An RO is delivered over an interaction channel. The encoding of the RO is specified in [DRMDRMv2.1].
Secure Authenticated Channel	A logical channel that provides message integrity and confidentiality.
Secure Removable Media	A removable media that implements means to protect against unauthorized access to its internal data and includes an SRM Agent. (e.g. secure memory card, smart card)
SRM Agent	A trusted entity embodied in Secure Removable Media. This entity is responsible for storing and removing Rights Objects in Secure Removable Media, for delivering Rights Objects from/to a DRM Agent in a secure manner, and for enforcing permissions and constraints, including securely maintaining state information for stateful rights. The SRM Agent is a part of Secure Removable Media.
SRM Rights Upgrade	To upgrade Rights on an SRM with additional permissions and constraints.
SRM to SRM Rights Move	To Move the Rights directly from one SRM to another SRM.
User	The human user of a Device. The User does not necessarily own the Device. (From [OMADRMv2.1])

3.3 Abbreviations

AES	Advanced Encryption Standard
BCAST	Mobile Broadcast Services
BCRO	Broadcast Rights Object
CBC	Cipher Block Chaining
CEK	Content Encryption Key
CRL	Certificate Revocation List
DER	Distinguished Encoding Rules
DRM	Digital Rights Management
HMAC	Keyed-Hash Message Authentication Code
HTTP	Hyper Text Transfer Protocol
IV	Initialisation Vector
KDF	Key Derivation Function
LAID	List of Asset Identifier
MAC	Message Authentication Code
MAKE	Mutual Authentication and Key Exchange
MK	MAC Key
OCSP	Online Certificate Status Protocol
OMA	Open Mobile Alliance

OMNA	Open Mobile Naming Authority
PKCS	Public Key Cryptography Standards
REK	Rights Object Encryption Key
REL	Rights Expression Language
RFC	Request For Comments
RI	Rights Issuer
RN	Random Number
ROID	Rights Object Identifier
RO	Rights Object
ROAP	Rights Object Acquisition Protocol
RSA	Rivest-Shamir-Adelman public key algorithm
RSA-OAEP	RSA encryption scheme - Optimal Asymmetric Encryption Padding
RSA-PSS	RSA Probabilistic Signature Scheme
R-UIM	Removable User Identity Module
SAC	Secure Authenticated Channel
SCR	Static Conformance Requirement
SD	Secure Digital
SHA1	Secure Hash Algorithm
SK	Session Key
S-MMC	Secure MultiMediaCard
SIM	Subscriber Identity Module
SRM	Secure Removable Media
URI	Uniform Resource Indicator
URL	Uniform Resource Locator
USIM	Universal Subscriber Identity Module
WBXML	Wireless Binary XML

3.4 Notations

The following notation is used in this specification:

X* | *Y	Concatenation of *X* and *Y*
E (*K* , *M*)	The result of encrypting message *M* using the RSA key *K*
H (*X*)	The result of computing a hash on *X*
HMAC (*K* , *X*)	The result of computing an HMAC on *X* using the key *K*

The following typographical conventions are used in the body of the text: `BinaryDataStructureVariables`, *Message Fields*, **<XML Elements>**

3.5 Binary Structures

This document uses a “C” like language to describe the binary data structures used. The details are provided in Appendix D.

4. Introduction

Secure Removable Media (SRM) is a removable media that implements the means to protect against unauthorised access to its internal data and includes an SRM Agent. Examples of SRM are the secure memory card and the smart card.

The secure memory card has an embedded microprocessor and is capable of storing Rights and DRM Content in a secure manner (e.g. S-MMC, SD). The smart card also has an embedded microprocessor and is capable of storing access codes, user subscription information, secret keys, DRM Content, Rights etc (e.g. SIM, USIM, R-UIM). Differently from the secure memory card, the smart card enables Users to make a telephone call by using the Devices and is issued by a mobile network operator.

If a User has a Device with a physical interface to an SRM, the User can use the SRM as means of increasing the storage space for DRM Content and for the portability of Rights.

The SRM Enabler provides a mechanism to write, read, delete and update Rights in SRM in a secure manner to realise the use cases defined in the OMA SRM requirements document [SRM-RDv1.1]. The architecture, security considerations, and trust model requirements for OMA SRM are specified in the OMA SRM architecture document [SRM-ADv1.1].

4.1 Version 1.0

While the OMA DRM version 2.0 [OMADRMv2.0] and version 2.1 [OMADRMv2.1] define an end-to-end system for DRM Content and Rights Object distribution among the Device, the Rights Issuer and the Content Issuer, SRM TS 1.0 defines mechanisms and protocols to extend OMA DRM version 2.0 to allow Users to Move Rights between the Device and the SRM and to consume Rights stored in the SRM. Especially, the definition of Rights format is compliant to OMA DRM REL 2.0 and 2.1.

4.2 Version 1.1

Based on the new Use Cases and requirements proposed in the SRM 1.1 RD, the SRM 1.1 TS defines several extensions to the SRM 1.0 TS. The security mechanism is consistent with SRM v1.0 and the definition of the Rights format is compliant to both the OMA DRM v2.1 and the SCE REL.

4.3 Component and Interface Deployment

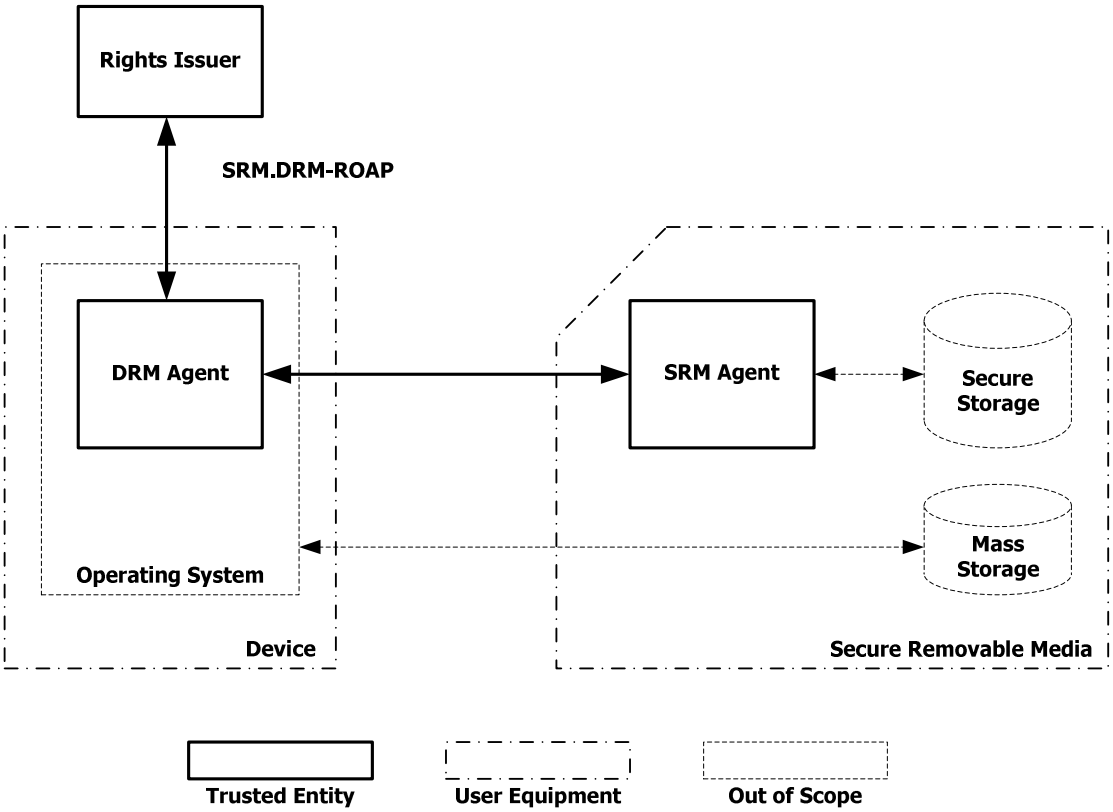

Figure 1: Secure Removable Media System - Component and Interface

The Secure Removable Media system is a set of three entities: Rights Issuer, DRM Agent and SRM Agent.

The Rights Issuer and DRM Agent communicate with each other by the ROAP protocol as defined in [OMADRMv2.0], [OMADRMv2.1], [SCE-DRM]. The DRM Agent and SRM Agent exchange messages as specified in section 6.

The SRM Agent has an internal secure communication with the secure storage. The implementation of the communication is out of scope of this specification. For the completeness of the security in the Secure Removable Media system, this specification assumes the follows:

- Only the SRM Agent can access the secure storage (i.e. the DRM Agent cannot directly access the secure storage).
- To perform an action on information in the secure storage, the DRM Agent requests the action to the SRM Agent. After performing the action, the SRM Agent passes the result of the action to the DRM Agent (i.e. the DRM Agent cannot receive information from the secure storage, if the information is not produced by the SRM Agent.).

5. Secure Removable Media Overview

This specification defines actions and interfaces of the Rights Issuer, DRM Agent, and SRM Agent.

5.1 Information Structure

5.1.1 Rights

This section specifies Rights exchanged with SRM(s). Rights may be stored in SRM(s) by being preloaded (refer to [SRM-AD v1.1]) or Moved from a Device by the Move Permission granted by Rights Issuers. Rights consist of Rights Meta Data, Rights Object Container, State Information and REK. XML elements and attributes referred to in this section are specified in [OMADRMv2.0].

The Rights MUST be securely stored in the SRM.

5.1.1.1 Rights Meta Data

Rights Meta Data consists of following information:

- Rights Object Version
- RO Alias
- RI Identifier
- RI URL
- RI Alias
- RI Time Stamp

Appendix E.2.5.1 specifies the data structure of the Rights Meta Data.

5.1.1.2 Rights Object Container

A Rights Object is a collection of Permissions and other attributes which are linked to DRM Content(s). The Rights Object is stored in an SRM in the format of the **Rights Object Container**. The SRM Agent treats the Rights Object Container as an opaque object, i.e. the SRM does not parse the Rights Object Container.

Consistent with the structure of a DRM 2.0 or 2.1 Rights Object, the Rights Object Container consists of the **<rights>** element and the **<signature>** element in the RO payload. The contents of the **<rights>** element of the Rights Object Container MUST be canonicalised as Exclusive Canonical XML format, as specified in [XC14N]. The RI-signature (i.e. **<signature>** element in the RO payload) MUST be present in the Rights Object Container. The RI-signature is created by a Rights Issuer that is identified by the **<riID>** element in the RO payload. The SRM Agent does not verify the RI-signature.

It is RECOMMENDED that the Rights Issuer not generate a Rights Object (in XML format) larger than 4096 bytes if the Rights Object may be stored in an SRM. Appendix E.2.5.2 specifies the data structure of the Rights Object Container.

DRM Agents MAY compact the Rights Object Container using WBXML (as defined in section 7) before transferring Rights from the Device to the SRM. The DRM Agent SHOULD compact the Rights Object Container if it is larger than 4096 bytes.

5.1.1.3 State Information

State Information is the current state (e.g. remaining counts, interval start date) of each stateful permission within a stateful Rights Object. This is present in Rights if the Rights Object is stateful. Appendix E.2.5.3 specifies the data structure of the State Information in detail.

5.1.1.4 REK

REK is Rights Object Encryption Key (REK) in binary form, i.e. no base64 encoding. Appendix E.2.5.5 specifies the data structure of the REK.

5.1.2 RI Certificate Chain

If the SRM Agent supports storage of RI Certificate Chains as indicated in the SrmHelloResponse, the DRM Agent SHOULD send to the SRM the RI Certificate Chains that are required to verify the signature of the Rights it transfers to an SRM Agent (refer to section 6.12.7).

A trust model's policy may determine whether the DRM Agent is required to verify the RI-signature when the Rights are installed in the Device as a part of the Rights Move (Refer to section 6.6). The default behaviour is that the DRM Agent MUST verify the RI-signature and its RI certificate chain.

When Rights in the SRM are used for the Local Rights Consumption (refer to section 6.7), the DRM Agent SHOULD verify the RI-signature.

If RI-signature verification is required and the SRM does not provide the RI certificate chain, the Device MUST get the certificate chain (if it does not have it already). The DRM Agent can acquire the RI certificate chain via a DRM v2.0 RI Registration or via other methods outside of ROAP. The DRM Agent is not required to check the RI revocation status and RI certification chain expiration during RI-signature verification.

5.1.3 Handle

The **Handle** is a random number generated by a DRM Agent and used to identify Rights on the SRM that the DRM Agent intends to access for the Move or Local Rights Consumption operation. The Handle is stored in the SRM and in the Operation Log of a Device.

When sending the initial message of Move or Local Rights Consumption, the DRM Agent MUST generate a Handle and send it to the SRM.

The usage of the Handle is specified in the sections for Move and Local Rights Consumption.

5.1.4 Rights Object Identifier

The **Rights Object Identifier** (ROID) uniquely identifies a Rights Object. This is the value of the **<uid>** element in the **<context>** element that is a child of the **<rights>** element in the Rights Object.

5.1.5 Asset Identifier

The **Asset Identifier** (AssetID) is included in a Rights Object and identifies a DRM Content. The identification may be equivalent to a subscription identifier or a Group ID for a corresponding group of DRM Contents, see [OMADRMv2.0]. When an RO contains Service Encryption Key (SEK) or Program Encryption Key (PEK) used by BCAST Enabler (see [SRM-ADv1.1]), the AssetID will be service_CID or program_CID defined in [DRMXBSv1.1].

Devices and SRMs MUST support AssetIDs of at least 256 bytes. It is RECOMMENDED that a content author not use an AssetID larger than 256 bytes if the Rights may be installed in an SRM.

5.1.6 Rights Information

Rights Information consists of Rights Meta Data, Rights Object Container, and State Information. This does not include the REK. The State Information is present in the Rights Information if the Rights Object in the Rights Object Container is stateful. Appendix E.2.5.4 specifies the data structure of the Rights Information.

5.1.7 List of Asset Identifier

The **List of Asset Identifier** (LAID) is a list that identifies DRM Content which is associated with a Rights Object. Appendix E.3 specifies the data structure of the LAID. The list comprises the hashes of AssetIDs.

5.1.8 Handle List

Handle List consists of the Handles of a group of enabled Rights in an SRM. Appendix E.4 specifies the data structure of the Handle List.

5.1.9 Rights Information List

Rights Information List consists of one or multiple Rights Information. Appendix E.2.6 specifies the data structure of the Rights Information List.

5.1.10 BCAST Tokens

The **BCAST Token Container** contains BCAST Tokens and is represented by a collection of metadata consisting of the following attributes:

- **RI ID**: Rights Issuer identifier which identifies Token issuer (authorizing RI). Appendix E.2.5 specifies data structure of this attribute.
- **Service ID/Program IDs**: one or more identifiers of services and/or programs for which the BCAST Tokens can be used.
- **Movable**: indicates whether the Token can be moved or not.
- **Domain IDs**: indicates that whether BCAST Token can be shared among members of the specific Broadcast Domain(s) defined in [DRMXBS 1.1].
- **Token Delivery ID**: identifies the BCAST Token in a similar fashion to the way an RO ID identifies a RO.
- **Reporting Information** is the compound attribute which consist of
    - **Token Reporting URL**: the presence of this parameter indicates that consumption of this BCAST Token must be reported. The parameter defines the URL to which the ROAPTokenConsumptionReport message should be sent.
    - **Latest Token Consumption Time**: after the date/time indicated in this parameter, the device SHALL NOT use any BCAST Tokens which have been received after the last ROAP-TokenDeliveryResponse message which includes a token reporting URL. If reports are being made on time by the device, this date is constantly being updated and therefore consumption should never be blocked. This field should only be defined when a token reporting URL is defined.
    - **Earliest Reporting Time**: the device should report consumption after this time and before the latest reporting time.
    - **Latest Reporting Time**: the device should report consumption before this time and after the earliest reporting time.
- **Token Quantity**: contains the number of BCAST Tokens in the BCAST Token Container.

Appendix E.2.10 specifies the data structures of the BCAST Token Container attributes.

5.1.11 Broadcast Rights

This section specifies Broadcast Rights exchanged with SRM(s). Broadcast Rights may be Moved from a Device if permission is granted by the Rights Issuer. Broadcast Rights consist of a Broadcast Rights Object (BCRO) Base, BCRO Assets, State Information and Signature.

Broadcast Rights MUST be securely stored in the SRM.

5.1.11.1 BCRO Base

BCRO Base is the OMADRMBroadcastRightsObjectBase() object defined in [DRMXBSv1.1]. It contains permissions and constraints associated to broadcast contents.

Appendix E.2.11 specifies the data structure of the BCRO Base.

5.1.11.2 BCRO Assets

BCRO Assets contains Program Encryption Authentication Key (PEAK), Service Encryption Authentication Key (SEAK) or Content Encryption Key (CEK), Asset Index and Binary Content Identifier (BCI) (see [DRMXBSv1.1] for more information).

- **PEAK and SEAK** are used to authenticate and encrypt BCAST traffic encryption keys (see [DRMXBSv1.1]).
- **CEK** is used for direct encryption of broadcast content.
- **Asset Index** is an internal identifier of the asset.
- **BCI is an identifier of associated broadcast (DRM) content.**

Appendix E.2.11 specifies the data structure of the BCRO Assets.

5.1.11.3 BCRO State Info

BCRO State Info contains state information if the BCRO is stateful.

Appendix E.2.11 specifies the data structure of the BCRO State Info.

5.1.11.4 BCRO Signature

BCRO Signature is an optional element and contains OMADRMBroadcastRightsObjectSigned() object excluding OMADRMBroadcastRightsObjectBase() object which is transferred and stored separately.

Appendix E.2.11 specifies the data structure of the BCRO Signature.

5.2 Security Algorithms

The following cryptographic algorithms are used in this specification. The following algorithms MUST be supported by all DRM Agents and SRM Agents.

Hash algorithms:

SHA-1 [SHA1]

MAC algorithms:

HMAC-SHA-1 [HMAC]

Symmetric encryption algorithms:

AES-128-CBC [AES]

Asymmetric encryption algorithms:

RSA-OAEP (v2.1) [PKCS-1]

Signature algorithms:

RSA-PSS (v2.1) [PKCS-1]

5.3 DRM Agent – SRM Agent Communications

A DRM Agent communicates to an SRM Agent over a physical communications channel. How this communication channel is established is beyond the scope of this document. It is presumed that the DRM Agent can use the services of the underlying operating system to discover and to establish the channel. Once this physical channel has been established, one or more logical channels can be established, depending on what kind of information needs to be exchanged and how many trust models are supported by the SRM Agent.

5.4 Client – Server Model

The model used for the communications between the DRM Agent and the SRM Agent is a client – server model. The DRM Agent is always the client and the SRM Agent is always the server. The SRM Agent does not act by itself. It only acts when it receives a request from a client (a DRM Agent) and then responds to that request.

In addition, it is always the DRM Agent that establishes the physical and logical communication channels.

5.5 Recovery Procedures

This section defines the process of exception handling.

5.5.1 Exception Handling

During the execution of access protocols between the DRM Agent and SRM Agent as specified in sections 6.5 to 6.10, exception handling may become necessary. Exceptional cases are, for example, the unexpected unplugging of the SRM or Device power-off. In particular, during Move or Local Rights Consumption various exceptions can occur that must be handled properly. The appropriate recovery steps are explained in the subsections of sections 6.5 to 6.10. This section defines an Operation Log and the Handle concept, which are needed for the recovery process.

This specification makes the following assumptions:

- If any exception occurs during the Local Rights Consumption in section 6.7, then the Device will stop using the associated DRM Content.
- In some cases, recovery may involve User interaction.

5.5.2 Operation Log

An **Operation Log** is a secure file, which MUST be kept by the Device, in which entries containing information about a transaction are stored until the transaction is completed. It is used for recovery procedures.

A “transaction” is a complete set of message pairs that must be exchanged between a DRM Agent and an SRM Agent in order to perform a particular SRM operation. The following transactions are defined:

- Movement of Rights from a Device to an SRM (section 6.5)
- Movement of Rights from an SRM to a Device (section 6.6)
- Local Rights Consumption (section 6.7)
- Direct Provisioning of Rights to the SRM (section 6.8)
- SRM Rights Upgrade (section 6.9)
- S2S Rights Move (section 6.10)

After a Secure Authenticated Channel is established as specified in section 6.2, the DRM Agent checks whether there is an entry in the Operation Log associated with the SRM Agent.

If any Operation Log entry exists, recovery procedures may be necessary. The DRM Agent analyses all Operation Log entries (associated with the SRM Agent) in order to determine appropriate recovery actions. Details on the recovery procedure are part of the description of each function in sections 6.5 to 6.10.

If no Operation Log entry exists, the DRM Agent MUST create an entry upon starting a Move or Local Rights Consumption transaction with the SRM as specified in sections 6.5 to 6.10. When a transaction is successfully completed, the DRM Agent SHOULD remove the entry.

An entry in the Operation Log is specified in Table 1.

Table 1: Operation Log Entry

Log	Description
SRM ID	This identifies an SRM that the Device is interacting with.
ROID	This identifies the Rights Object that is the target of a transaction. Refer to section 5.1.4
Handle	This is generated by the DRM Agent and identifies Rights in the SRM.
Transaction Identifier	This identifies the transaction that the DRM Agent and SRM Agent are performing. This field is fixed for the duration of the transaction. It SHALL be possible to use this field to determine which entry is the oldest in the Operation Log.
Current Step	This represents the current execution step of a transaction as identified by the function identifier and also records whether the transaction is successfully completed or not. If the function identifier refers to "Movement of Rights from Device to SRM", then the DRM Agent makes a record after starting one of the following steps: • Installation Setup Message • Rights Disablement in Device • Rights Installation Message • Rights Removal in Device If the function identifier refers to "Movement of Rights from SRM to Device", then the DRM Agent makes a record after starting one of the following steps: • Rights Retrieval Message • Rights Installation in Device • Rights Removal Message If the function identifier refers to "Local Rights Consumption", then the DRM Agent makes a record after starting one of the following steps: • REK Query Message • Rights Enablement Message If the function identifier refers to "Direct Provisioning of Rights to the SRM", then the DRM Agent makes a record after starting one of the following steps: • Signature Query Message • Provisioning Setup Message • Rights Provisioning Message • Rights Removal Message If the function identifier refers to "SRM Rights Upgrade", then the DRM Agent makes a record after starting one of the following steps:

Log	Description
	• Upgrade Rights Retrieval • RO Upgrade If the function identifier refers to "S2S Rights Move", then the DRM Agent makes a record after starting one of the following steps: • S2S Move Initiation • Installation Setup • Rights Installation • Rights Removal

If the Operation Log is full, before a new entry is added, the oldest entry, based on the Transaction Identifier, SHALL be removed. Note that removing an entry will prevent any recovery procedure associated with the entry. While the size of the Operation Log is not specified in this document, it should be large enough to minimise the effect of removing entries for incomplete transactions.

5.5.3 Operation Log Extensions for BCAST

The operations described in section 5.5.2 apply to SRM extensions for BCAST. This section defines necessary extensions to support the following transactions and procedures:

- Movement of BCAST Tokens from Device to SRM (section 6.11.1)
- Movement of BCAST Tokens from SRM to Device (section 6.11.2)
- Local BCAST Token Consumption (section 6.11.3)
- BCAST Token Upgrade on the SRM procedure (section 6.11.6)
- Movement of BCRO from Device to SRM (section 6.11.7)
- Movement of BCRO from SRM to Device (section 6.11.8)
- Local BCAST Rights Consumption (section 6.11.9)

The DRM Agent MUST create a Log entry upon starting any of these transactions. When a transaction is successfully completed, the DRM Agent SHOULD remove the entry.

The DRM Agent that supports Token related operations SHALL use Extended Operation Log.

An entry in the Extended Operation Log is specified in Table 2.

Table 2: Extended Operation Log Entry

Log	Description
Entry Type	This indicates a type of log entry. Two types are currently defined: - Rights Object Transaction Log Entry - BCAST Token Transaction Log Entry
Entry Contents	If *Entry Type* field indicates "Rights Object Transaction Log Entry" contents of this field will be operation log entry defined in section 5.5.2. If *Entry Type* field indicates "BCAST Token Transaction Log Entry" contents of this field are defined in Table 3.

Table 3: BCAST Token Transaction Log Entry

Log	Description
SRM ID	This identifies an SRM that the Device is interacting with.
RI ID	This identifies the Rights Issuer that issued the BCAST Token under consideration
Token Delivery ID	This together with RI ID uniquely identifies BCAST Tokens/BCAST Token Container stored on the SRM.
Transaction Identifier	This identifies the transaction that the DRM Agent and SRM Agent are performing. This field is fixed for the duration of the transaction. It SHALL be possible to use this field to determine which entry is the oldest in the Operation Log.
Current Step	This represents the current execution step of a transaction as identified by the function identifier and also records whether the transaction is successfully completed or not. If the function identifier refers to "Movement of BCAST Tokens from Device to SRM", then the DRM Agent makes a record after starting one of the following steps: • BCAST Token Disablement in Device • BCAST Token Installation Request • BCAST Token Installation Response If the function identifier refers to "Movement of BCAST Tokens from SRM to Device", then the DRM Agent makes a record after starting one of the following steps: • BCAST Token Retrieval Request • BCAST Token Installation in Device • BCAST Token Removal Request If the function identifier refers to "Local Token Consumption", then the DRM Agent makes a record after starting one of the following steps: • BCAST Token Consumption Request • BCAST Token Enablement Request If the function identifier refers to "BCAST Token Upgrade", then the DRM Agent makes a record after starting one of the following steps: • BCAST Token Upgrade Request • BCAST Token Disablement in Device

Operation Log Entry specified in section 5.5.2 SHALL support the following transactions:

- Movement of BCRO from Device to SRM (section 6.11.7)
- Movement of BCRO from SRM to Device (section 6.11.8)
- Local BCAST Rights Consumption (section 6.11.9)

In order to support these transactions, the following extensions are defined:

- If the function identifier refers to "Movement of BCRO from Device to SRM", then the DRM Agent MUST make a record after starting one of the following steps:
  - Installation Setup Message
  - BCRO Asset Disablement in Device
  - Broadcast Rights Installation Message

    - Broadcast Rights Removal in Device
- If the function identifier refers to “Movement of BCRO from SRM to Device”, then the DRM Agent MUST make a record after starting one of the following steps:
    - Broadcast Rights Retrieval Message
    - Broadcast Rights Installation in Device
    - Broadcast Rights Removal Message
- The function identifier of Local BCAST Rights Consumption transaction MUST equal to the function identifier of Local Rights Consumption transaction.

5.6 Notations of Messages

This section presents notations used in this specification.

5.6.1 Messages

A message is data sent between a DRM Agent and an SRM Agent in this specification. The communication is based on a request-response mechanism, e.g. first the DRM Agent sends a request message, and the SRM Agent processes the message and then sends back a response with the results of processing the request.

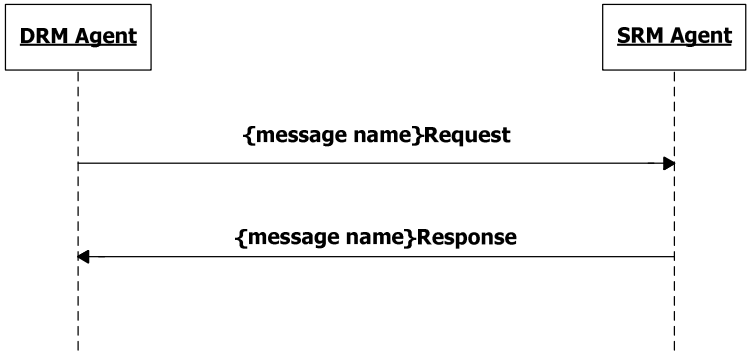

Figure 2: Notation of Message

In Figure 2, the solid line from the DRM Agent to the SRM Agent denotes a request and the solid line from the SRM Agent to the DRM Agent denotes a response. The DRM Agent sends the request to the SRM Agent to perform a specific action. After this, the SRM Agent sends the response back to the DRM Agent.

Names of requests and responses are ended with the string “Request” and “Response” (e.g. ***RightsInstallationRequest*** and ***RightsInstallationResponse***).

This notation is used for all messages in this specification.

5.6.2 Actions

An action is a specific operation of the DRM Agent or the SRM Agent. The DRM Agent performs a specific action independently, but the SRM Agent performs a specific action by a request from the DRM Agent. For each action in the SRM, the SRM Agent sends a response to the DRM Agent.

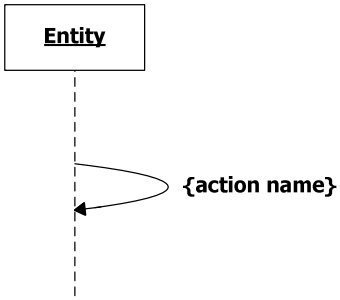

Figure 3: Notation of Action

In Figure 3, the curved line denotes an action. The entity (DRM Agent or SRM Agent) performs an action. Each action has a name (e.g. **RightsInstallationInSRM**). All action names are ended with the string "**In {Place}**", in case that the action is performed in the "**Place**".

5.6.3 Fields

A field is a data unit within a message which is passed from an entity to the other entity to make the recipient entity perform an action based on the value of the field.

Messages in this specification carry a set of fields from the DRM Agent to the SRM Agent or vice versa. The fields are denoted by using a table as Table 4 below. A request and response have their own field tables (i.e. one field table for the request and one field table for the response).

Table 4: Notation of Fields

Fields	Protection Requirement	Description
A	Integrity	
B	Confidentiality	
C	Integrity & Confidentiality	
D	No	

The Table 4 shows that a message carries 4 fields – A, B, C, and D. The "**Protection Requirement**" column denotes the security properties provided by the Secure Authenticated Channel. The "**Description**" column describes the fields.

5.6.4 Message Format

Some messages between the DRM Agent and SRM Agent are protected by an HMAC and some are not. See Table 5 for which messages are protected by an HMAC.

Messages between the DRM Agent and SRM Agent that are not protected by an HMAC have the following generic format:

```
MessageFormat ()
    messageIdentifier        7      bslbf
    messageType              1      bslbf
    MessageBody()
    ExtensionsContainer()
}
```

Messages between the DRM Agent and SRM Agent that are protected by an HMAC have the following generic format:

```
ProtectedMessageFormat() {
    MessageFormat()
    Hmac()                          // Defined in Appendix E.1
}
```

The fields are defined as follows:

- **`messageIdentifier`** - This field defines the identifier of messages being communicated. This is defined in Table 5
- **`messageType`** - This flag is set to '0' if this is a request from the DRM Agent to SRM Agent. In case of a response, it is set to '1'.
- **`MessageBody`** - This field contains fields of a message. The **`MessageBody`** is specified in each sub-section in section 6.
- **`ExtensionsContainer`** - This field can be used to include extensions in future versions of the SRM enabler. See section 5.6.5 for more details.
- **`Hmac`** - HMAC over **`MessageFormat`**, generated with the current MAC Key (MK). This field only exists for messages that are integrity protected by an HMAC. Table 5 indicates which messages are protected by an HMAC.

Table 5: Message Identifier

Identifier Value	Description	Protection		Mandatory/Optional	
		Request protected by an HMAC	Response protected by an HMAC	Support by DRM Agent	Support by SRM Agent
0	SRM Hello	NO	NO	M	M
1	Authentication	NO	NO	M	M
2	Key Exchange	NO	NO	M	M
3	CRL Information Exchange	NO	NO	M	M
4	OCSP Nonce	NO	NO	O	O
5	OCSP Process	NO	NO	O	O
6	CRL Update	NO	NO	M	M
7	CRL Retrieval	NO	NO	M	M
8	Installation Setup	YES	YES	M	M
9	Rights Installation	YES	YES	M	M
10	Rights Retrieval	YES	YES	M	M
11	REK Query	YES	YES	M	M
12	Rights Info Query	YES	YES	M	M
13	Handle List Query	NO	NO	M	M
14	Handle Removal	YES	YES	M	M
15	Rights Enablement	YES	YES	M	M
16	Rights Removal	YES	YES	M	M
17	RI Certificate Store	NO	NO	O	O
18	RI Certificate Query	NO	NO	O	O
19	RI Certificate Removal	NO	NO	O	O
20	Dynamic Code Page Query	NO	NO	M	O
21	Dynamic Code Page Update	NO	NO	O	O
22	Rights Info List Query	NO	YES	O	O
23	Change SAC	NO	NO	O	O

Identifier Value	Description	Protection		Mandatory/Optional	
		Request protected by an HMAC	Response protected by an HMAC	Support by DRM Agent	Support by SRM Agent
24	Signature Query	YES	YES	M	M
25	Provisioning Setup	YES	YES	M	M
26	Rights Provisioning	YES	YES	M	M
27	Upgrade Rights Retrieval	YES	YES	M	M
28	Rights Upgrade	YES	YES	M	M
29	S2S Move Initiation	YES	YES	M	M
30	BCAST Token Installation	YES	YES	O	O
31	BCAST Token Retrieval	YES	YES	O	O
32	BCAST Token Removal	YES	YES	O	O
33	BCAST Token Consumption	YES	YES	O	O
34	BCAST Token Enablement	YES	YES	O	O
35	BCAST Token Information	YES	YES	O	O
36	BCAST Token Upgrade	YES	YES	O	O
37	Broadcast Rights Installation	YES	YES	O	O
38	Broadcast Rights Retrieval	YES	YES	O	O
39	SRM Ping	YES	YES	O	O
40	BCRO Information Retrieval	YES	YES	O	O
41	BCRO Information List Retrieval	YES	YES	O	O
42 ~ 127	Reserved For Future Use				

In Table 5, ‘M’ denotes that the DRM or SRM Agents MUST support the messages, and ‘O’ denotes that the agents MAY support the messages.

5.6.5 Extensibility of Binary Messages

All messages between a DRM Agent and an SRM Agent contain an **`ExtensionsContainer()`** structure. This structure has the following format:

```
ExtensionsContainer() {
    nbrOfExtensions          8      uimsbf
    for( i = 0 ; i < nbrOfExtensions ; i++ ) {
        extensionType        8      uimsbf
        size                16      uimsbf
        Extension()
    }
}
```

The **`nbrOfExtensions`** field indicates how many extensions follow

Each extension the **`ExtensionsContainer()`** structure contains the following fields:

- **`extensionType`** - an 8-bit integer signalling the type of the extension. Each extension SHALL have a unique extensionType. Table 7 specifies the values of the extensionType.
- **`size`** - a 16-bit integer specifying the size of the extension, i.e. the size of the **`Extension`** field in bytes. If the receiver of the message does not know the extension type, this field can be used to skip to the next extension.
- **`Extension`** - this structure contains the fields of the extension. The content of the structure depends on the particular extension and is to be defined in future specifications.

Table 6: extensionType Values

Value	Description
0	BCAST extension to Handle List Query (see Section 6.11.10.1)
1	BCAST extension to Rights Enablement (see Section 6.11.10.4)
2-255	Reserved

Unknown extensions SHALL be ignored by the receiving (DRM or SRM) Agent.

5.6.5.1 Application of Extensibility in Future Specifications (informative)

Future specifications can use the **`ExtensionsContainer`** field within the **`MessageFormat`** structure to expand messages. When an extension is specified in a future specification, it can either be included in all messages independent of the version of the involved SRM/DRM Agents or only included when communication between agents of appropriate versions occurs. The decision on where and when a certain extension is to be included is to be taken when the new specification is made.

Extensions can be mandated in future specifications. This means DRM/SRM Agents conformant to those specifications must include the extensions, even though older SRM/DRM Agents will ignore it. The extensions have to be designed in such a way that this does not open an attack opportunity.

In SRM v1.1, extensibility of the **`MetaData`**, **`PermissionState`** and **`ConstraintState`** structures has been specified by including an ExtensionsContainer field (see section E.2.5.1 and E.2.5.3).

Even though SRM v1.0 supports the **`length`** field in the **`RightsMetaData`** structure, it is not specified if and how the DRM and SRM Agents behave when it receives an extended **`MetaData`** structure. Therefore, using **`MetaData`** extensions in the future could lead to incompatibilities with SRM v1.0.

A similar reasoning holds for the **`PermissionState`** and **`ConstraintState`** structures, although these structures do not have an associated **`length`** field. Using the **`ExtensionsContainer`** field in these structures would also lead to incompatibilities with SRM v1.0.

Since SRM v1.0 supports the **`ExtensionsContainer`** in the **`MessageFormat`** structure, it is recommended to use that extensibility option when possible. However, if extension of the **`MetaData`**, **`PermissionState`** and **`ConstraintState`** structures cannot be done through extending the **`MessageFormat`** structure, it is best to use the **`ExtensionsContainer`** mechanism in **`MetaData`**, **`PermissionState`** and **`ConstraintState`** structures. Extending the structures in another way will not only result in incompatibility with SRM v1.0, but also with SRM v1.1. Using the **`ExtensionsContainer`** mechanism at least guarantees backward compatibility with SRM v1.1.

5.6.6 Status

Each response (i.e. **`messageType`** is set to 1) has a **`Status`** field (see Appendix E.2.2) indicating whether its corresponding request (i.e. **`messageType`** is set to 0) was successfully processed or not. Table 7 lists the integer values assigned to each status code.

Table 7: Status Code Values

Value	Status Name
0	Success
1	Unknown Error
2	Trust Anchor Not Supported
3	Device Certificate Chain Verification Failed
4	Field Decryption Failed
5	SRM Random Number Mismatched

Value	Status Name
6	Version Mismatched
7	SAC Not Established
8	Old CRL
9	OCSP Response Verification Failed
10	Invalid OCSP Nonce
11	CRL Verification Failed
12	CRL Not Found
13	Field Integrity Verification Failed
14	Duplicate Handle
15	Not Enough Space
16	Handle Not Found
17	Handle Not Removed
18	Request Not Supported
19	RI Certificate Chain Not Found
20	Dynamic Code Pages Not Found
21	Handles In-consistent
22	Parameter Failed
23	Unexpected Request
24	AssetID List Too Long
25	Signature Scheme Not Supported
26	Storage Reservation Failed
27	Disablement Failed
28	Insufficient Amount of Tokens
29	Invalid Token Amount
30	Token Not Found
31	Remove Failed
32	Not an RORequest
33 ~ 65535	Reserved For Future Use

In section 6, for each description of a request/response message pair, a list of valid status values is specified. Should a DRM Agent receive a status value not specified for a particular response message, the DRM Agent SHALL treat the status as having received *Unknown Error*.

SRM Agent MUST return “Request Not Supported” Status Code in case it receives a request message with Message Identifier of “Reserved For Future Use” (defined in Table 5) or if the request is not supported. The Message Identifier of the response SHALL be equal to the Message Identifier of the request. The response MUST only contain **`Status`** field and SHALL be protected by an HMAC.

6. DRM Agent – SRM Agent Protocol

6.1 SRM Hello

The SRM Hello message pair is used by the DRM Agent and the SRM to exchange information about each other.

6.1.1 Hello

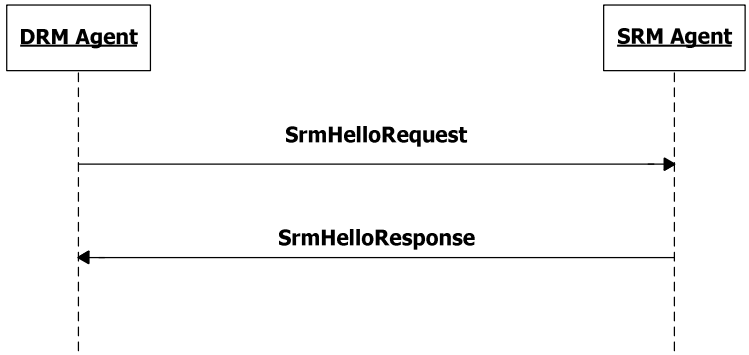

Figure 4: Sequence Diagram – SRM Hello

6.1.1.1 Description of Messages

The DRM Agent sends the SrmHelloRequest to initiate a logical channel with the SRM Agent. The fields of the request are defined in Table 8.

Table 8: Fields of SrmHelloRequest

Fields	Protection Requirement	Description
Version	No	*Version* is a <major.minor> representation of the highest SRM protocol version number supported by the DRM Agent. For this version of the protocol, *Version* SHALL be set to 1.1.
Trust Anchor And Device ID Pair List	No	*Trust Anchor And Device ID Pair List* contains the list of trust anchor and Device ID pairs for the Device. The trust anchor identifies the trust model. If the Device has more than one Device ID under a trust model, then only one Device ID under the trust model MUST be present in this list.

Upon receiving the SrmHelloRequest, the SRM Agent selects a protocol version supported by the SRM.

The SRM Agent checks to see if it supports any of the trust anchors in the *Trust Anchor And Device ID Pair List*. If not, set *Status* to *Trust Anchor Not Supported* and send the SrmHelloResponse.

After completing this step, the SRM Agent sends the SrmHelloResponse to the DRM Agent. The fields of the response are defined in Table 9.

Table 9: Fields of SrmHelloResponse

Fields	Protection Requirement	Description

Fields	Protection Requirement	Description
Status	No	The result of processing the SrmHelloRequest message. The *Status* values are specified in Table 10. If *Status* contains any error, only this field is present in the SrmHelloResponse.
Selected Version	No	The protocol version selected by the SRM Agent. The *Selected Version* will be *min*(DRM Agent suggested version, highest version supported by the SRM Agent). The *min*(A,B) = A where A <= B.
Trust Anchor And SRM ID Pair List	No	*Trust Anchor And SRM ID Pair List* contains the list of trust anchor and SRM ID pairs for the SRM. The trust anchor identifies the trust model. The trust anchors MUST be one of the trust anchors in the *Trust Anchor And Device ID Pair List* in the SrmHelloRequest. For example, if the *Trust Anchor And Device ID Pair List* has trust anchors A, B and C and the SRM supports trust anchors B, C and D, then the *Trust Anchor And SRM ID Pair List* would only contain trust anchors B and C. If the SRM has more than one SRM ID under a trust model, then only one SRM ID under the trust model MUST be present in this list.
Peer Key Identifier List	No	*Peer Key Identifier List* contains a list of Device IDs stored by the SRM. If any of the identifiers match the Device IDs in the *Trust Anchor and Device ID Pair List* in the preceding SrmHelloRequest, it means the SRM has already verified the corresponding Device's certificate chains, and that the DRM Agent does not need to send any of those certificate chains in a later message. If the SRM has verified the Device's certificate chain, based on the *Trust Anchor and Device ID Pair List* in the SrmHelloRequest, then the SRM Agent MUST include this field in the SrmHelloResponse.
Max Number Of AssetIDs	No	This field contains the maximum number of H(AssetID)s that can be processed by the SRM Agent in the HandleListQueryRequest (see section 6.12.1.1).
Optional Messages Supported	No	This field indicates which optional messages are supported by the SRM.

Table 10: Status of Srm Hello Message

Status Value	Description
Success	The request was successfully processed
Trust Anchor Not Supported	*Trust Anchor* in the request is not supported by the SRM Agent
Parameter Failed	A field in the request has an invalid length or structure.
Unknown Error	Other errors

Upon receiving the SrmHelloResponse and *Status* is *Success*, the DRM Agent continues with the MAKE process in section 6.2.

6.1.1.2 Format of Messages

The message format (**`MessageBody`**) of the SrmHelloRequest is specified as follows. The **`messageType`** is set to '0' and the message is not protected by an HMAC.

```
EntityId() {
    // SHA-1 of the DER-encoded
    // subjectPublicKeyInfo component
    // of the Entity's certificate
    Hash()                          // Defined in Appendix E.1
}


DeviceId() {
    EntityId()
}


TrustAnchor(){
    // SHA-1 of root public key
    EntityId()
}


TrustAnchorAndDeviceIdPairList() {
    // There MUST be at least one pair
    nbrOfPairs               8      uimsbf
    for ( i = 0 ; i < nbrOfPairs ; i++ ) {
        TrustAnchor()
        // The Device's ID under the Trust Anchor above
        DeviceId()
    }
}


MessageBody() {
    Version()                       // Defined in Appendix E.2.1
    TrustAnchorAndDeviceIdPairList()
}
```

The fields are defined as follows:

- **`Version`** - *Version* field in Table 8
- **`TrustAnchorAndDeviceIdPairList`** – *Trust Anchor and Device ID Pair List* field in Table 8

The message format (**`MessageBody`**) of the SrmHelloResponse is specified as follows. The **`messageType`** is set to '1' and the message is not protected by an HMAC.

```
SelectedVersion() {
    Version()                       // Defined in Appendix E.2.1
}


SrmId(){
    EntityId()
}
```

```
TrustAnchorAndSrmIdPairList() {
    // There MUST be at least one pair
    nbrOfPairs                      8   uimsbf
    for ( i = 0 ; i < nbrOfPairs ; i++ ) {
        TrustAnchor()
        // The SRM's ID under the Trust Anchor above
        SrmId()
    }
}


PeerKeyIdentifier() {
    EntityId()
}


PeerKeyIdentifierList() {
    nbrOfPeerKeyIdentifiers         8   uimsbf
    for ( i = 0 ; i < nbrOfPeerKeyIdentifiers ; i++ ) {
        PeerKeyIdentifier()
    }
}


OptionalMessages() {
    ocspSupported                       1           bslbf
    rightsInfoListSupported             1           bslbf
    riCertificateStorageSupported       1           bslbf
    riCertificateRemovalSupported       1           bslbf
    dynamicCodePageSupported            1           bslbf
    changeSacSupported                  1           bslbf
    rfu                                 10          bslbf
}


MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        SelectedVersion()
        TrustAnchorAndSrmIdPairList()
        PeerKeyIdentifierList()
        maxNbrOfAssetIds                16          uimsbf
        OptionalMessages()
    }
}
```

The fields are defined as follows:

- **`Status`** - *Status* field in Table 9
- **`SelectedVersion`** - *Selected Version* field in Table 9
- **`TrustAnchorAndSrmIdPairList`** – *Trust Anchor And SRM ID Pair List* field in Table 9
- **`PeerKeyIdentifierList`** – *PeerKeyIdentifierList* field in Table 9
- **`maxNbrOfAssetIds`** – *Max Number of AssetIDs* field in Table 9
- **`OptionalMessages`** – *Optional Messages Supported* field in Table 9. The contained flags have meaning as follows:

- **ocspSupported** – if '0', the OCSP Nonce and OCSP Process messages in section 6.4.2 and 6.4.3 are not supported by the SRM Agent. If '1', the messages are supported by the SRM Agent.
- **rightsInfoListSupported** – if '0', the Rights Info List Query message in section 6.12.3 is not supported by the SRM Agent. If '1', the message is supported by the SRM Agent.
- **riCertificateStorageSupported** – if '0', the RI Certificate Store and RI Certificate Query messages in section 6.12.7 and 6.12.8 are not supported by the SRM Agent. If '1', the messages are supported by the SRM Agent.
- **riCertificateRemovalSupported** – if '0', the RI Certificate Removal message in section 6.12.9 is not supported by the SRM Agent. If '1', the message is supported by the SRM Agent.
- **dynamicCodePageSupported** – if '0', the Dynamic Code Page Query and Dynamic Code Page Update messages in section 6.12.10 and section 6.12.11 are not supported by the SRM Agent. If '1', the messages are supported by the SRM Agent.
- **changeSacSupported** – if '0', the ChangeSac messages in section 6.3.5 are not supported by the SRM Agent. If '1', the messages are supported by the SRM Agent.

6.1.1.3 Exception Handling

There may be an unexpected exception during the Srm Hello message pair processing as specified in section 5.5.1. If the DRM Agent fails to receive the response or finds an error by referring to the *Status*, then the DRM Agent regards it as an exception and terminates communication by discarding any existing SAC context with the SRM Agent. The User may be informed of this exception.

6.2 MAKE (Mutual Authentication and Key Exchange) Process

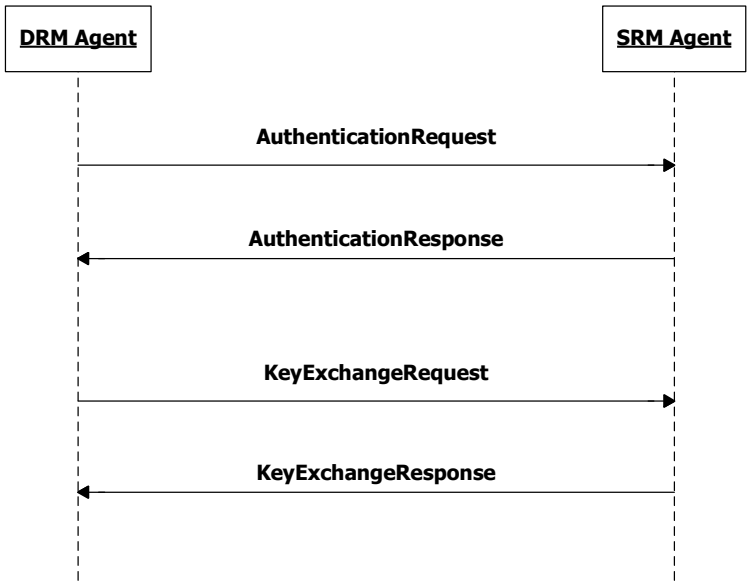

Figure 5: Sequence Diagram – MAKE Process

As shown in Figure 5, the MAKE Process is comprised of two request/response message pairs: Authentication message pair and Key Exchange message pair. The Authentication message pair SHOULD be followed by the Key Exchange message pair. If an SRM Agent receives any request message other than KeyExchangeRequest following receipt of the AuthenticationRequest, the SRM Agent SHOULD return *Unexpected Request* in the *Status* field of the response message.

6.2.1 Authentication

The DRM Agent sends the AuthenticationRequest to the SRM Agent to start the MAKE process. This request expresses Device information and preferences. The AuthenticationResponse expresses SRM information and preferences. The DRM Agent and SRM Agent may also exchange their certificate chains and verify them.

6.2.1.1 Description of Messages

The DRM Agent sends the AuthenticationRequest to initiate a MAKE process. The fields of the request are defined in Table 11.

Table 11: Fields of AuthenticationRequest

Fields	Protection Requirement	Description
Trust Anchor	No	*Trust Anchor* preferred by the DRM Agent. The trust anchor MUST be selected from *Trust Anchor And SRM ID Pair List* in the SrmHelloResponse. Selection of the trust anchor implicitly selects both the Device ID and the SRM ID.
Device Certificate Chain	No	A certificate chain for the Device under the selected trust anchor. The chain MUST NOT include the root certificate. The Device Certificate MUST come first in the list. Each following certificate MUST directly certify the one preceding it. Refer to Appendix H.1 If the *Peer Key Identifier List* field is present in the SrmHelloResponse and the list contains the Device ID corresponding to the Device Certificate Chain, then this field need not be sent in the AuthenticationRequest.
Peer Key Identifier	No	An SRM ID under the trust anchor indicated by the *Trust Anchor* field in this message. If the Device has already verified the corresponding SRM Certificate Chain, then this field SHOULD be present. This informs the SRM to not send the SRM's certificate chain in the AuthenticationResponse.
Supported Algorithms	No	*Supported Algorithms* identifies the cryptographic algorithms (hash algorithms, MAC algorithms, signature algorithms, asymmetric encryption algorithms, symmetric encryption algorithm, and key derivation functions) that are supported by the DRM Agent. Use of algorithms not listed in section 5.2 and 6.3.1 is optional. Since all DRM Agents and all SRM Agents must support the default algorithms, they need not be sent in this field. Only identifiers for algorithms that are not one of the defaults need to be sent in the AuthenticationRequest.

Upon receiving the AuthenticationRequest, the SRM Agent MUST perform the following procedure:

1. Check if it supports the *Trust Anchor*. If not, set *Status* to *Trust Anchor Not Supported* and send the AuthenticationResponse.

2. If present, verify the *Device Certificate Chain*. If the verification is good, then continue with step 5. Otherwise, set *Status* to *Device Certificate Chain Verification Failed* and send the AuthenticationResponse.

3. If the *Device Certificate Chain* is not present, do the following:

A. If the SrmHelloResponse did not include the *Peer Key Identifier List*, set *Status* to *Device Certificate Chain Verification Failed* and send the AuthenticationResponse.

B. If the SrmHelloResponse did include the *Peer Key Identifier List*, then check whether the *Trust Anchor* matches any trust anchor in the *Peer Key Identifier List*. If it does not, then set *Status* to *Device Certificate Chain Verification Failed* and send the AuthenticationResponse.

4. Check the *Peer Key Identifier* and determine whether or not to send the SRM's certificate chain under the *Trust Anchor*.

5. Select the algorithms to use from the *Supported Algorithms*.

After these steps, the SRM Agent sends the AuthenticationResponse to carry the result of the action. The fields of the response are defined in Table 12.

Table 12: Fields of AuthenticationResponse

Fields	Protection Requirement	Description
Status	No	The result of processing the AuthenticationRequest message. The *Status* values are specified in Table 13. If *Status* contains any error, only this field is present in the AuthenticationResponse.
SRM Certificate Chain	No	The SRM's certificate chain under the trust anchor sent in the preceding request. The chain MUST NOT include the root certificate. The SRM Certificate MUST come first in the list. Each following certificate MUST directly certify the one preceding it. Refer to Appendix H.1. If the *Peer Key Identifier* field was present in the preceding request, then this field SHOULD NOT be present.
Encrypted AuthResp Data	No	E ($PuKey_D$, AuthRespData) where AuthRespData = RN_S | *Version* | *Selected Algorithms* | H(*Supported Algorithms*) RN_S is a random number generated by the SRM Agent. *Version* is copied from the *Version* field in the SrmHelloRequest. *Selected Algorithms* specifies the cryptographic algorithms selected by the SRM Agent. H(*Supported Algorithms*) is the hash, using the selected hash algorithm, of the *Supported Algorithms* field in the AuthenticationRequest. *AuthRespData* is encrypted with the Device's public key ($PuKey_D$) under the trust anchor specified in the AuthenticationRequest.

Table 13: Status of Authentication Message

Status Value	Description
Success	The request was successfully processed.
Trust Anchor Not Supported	*Trust Anchor* in the request is not supported by the SRM Agent
Device Certificate Chain Verification Failed	The SRM Agent failed to verify the *Device Certificate Chain*.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the AuthenticationResponse and *Status* is *Success*, the DRM Agent verifies the *SRM Certificate Chain* if the certificate chain is present. If the DRM Agent did not send a *Peer Key Identifier* in the AuthenticationRequest and the certificate chain is not present, then the DRM Agent MUST terminate communications with the SRM. After the verification, the DRM Agent decrypts RN_S, *Version*, *Selected Algorithms*, and H(*Supported Algorithms*) with the Device's private key (under the trust anchor sent in the AuthenticationRequest).

The DRM Agent compares *Version* to the *Version* field sent in the SrmHelloRequest, and validates that it supports the *Selected Algorithms*. If the *Selected Algorithms* are not supported, then the DRM Agent MUST terminate communications with the SRM. Otherwise, using the selected hash algorithm, the DRM Agent validates the H(*Supported Algorithms*). If valid, the DRM Agent continues with section 6.2.2.

6.2.1.2 Format of Messages

The message format (**`MessageBody`**) of the AuthenticationRequest is specified as follows. The **`messageType`** is set to '0' and the message is not protected by an HMAC.

```
DeviceCertificateChain() {
    CertificateChain()              // Defined in Appendix E.1
}


AlgorithmList() {
    // If number of algorithms is zero,
    // then the default algorithm is used
    nbrOfAlgorithms          8      uimsbf
    for ( i = 0 ; i < nbrOfAlgorithms ; i++ ) {
        Algorithm()                 //Defined in Appendix E.1
    }
}


SupportedAlgorithms() {
    // Hash algorithms
    AlgorithmList()
    // HMAC algorithms
    AlgorithmList()
    // Symmetric algorithms
    AlgorithmList()
    // Asymmetric algorithms
    AlgorithmList()
    // KDF algorithms
    AlgorithmList()
}


MessageBody() {
    TrustAnchor()                   // Defined in section 6.1.1.2
    DeviceCertificateChain()
    PeerKeyIdentifier()             // Defined in section 6.1.1.2
    SupportedAlgorithms()
}
```

The fields are defined as follows:

- **`TrustAnchor`** – *Trust Anchor* field in Table 11
- **`DeviceCertificateChain`** – *Device Certificate Chain* field in Table 11
- **`PeerKeyIdentifier`** – *Peer Key Identifier* field in Table 11

- **SupportedAlgorithms** – *Supported Algorithms* field in Table 11

The message format (**MessageBody**) of the AuthenticationResponse is specified as follows. The **messageType** is set to '1' and the message is not protected by an HMAC.

```
SelectedAlgorithms() {
    // Hash algorithms
    AlgorithmList()
    // HMAC algorithms
    AlgorithmList()
    // Symmetric algorithms
    AlgorithmList()
    // Asymmetric algorithms
    AlgorithmList()
    // KDF algorithms
    AlgorithmList()
}


HashOfSupportedAlgorithms() {
    // Hash of SupportedAlgorithms from the
    // AuthenticationRequest, using the
    // hash from SelectedAlgorithms
    Hash()                          // Defined in Appendix E.1
}


SrmCertificateChain() {
    CertificateChain()              // Defined in Appendix E.1
}


AuthRespData() {
    RandomNumber()                  // Defined in Appendix E.1
    Version()                       // Defined in Appendix E.2.1
    SelectedAlgorithms()
    HashOfSupportedAlgorithms()
}


EncryptAuthRespData() {
    // Contains the encrypted AuthRespData
    EncryptedData()                 // Defined in Appendix E.1
}


MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        SrmCertificateChain()
        EncryptedAuthRespData()
    }
}
```

The fields are defined as follows:

- **AuthRespData** – *AuthRespData* value of *Encrypted AuthResp Data* field in Table 12

- **`RandomNumber`** – RN_S value of *Encrypted AuthResp Data* field in Table 12
- **`Version`** –*Version* value of *Encrypted AuthResp Data* field in Table 12
- **`SelectedAlgorithms`** – *Selected Algorithms* value of *Encrypted AuthResp Data* field in Table 12
- **`HashOfSupportedAlgorithms`** – H(*Supported Algorithms*) value of *Encrypted AuthResp Data* field in Table 12
- **`Status`** - *Status* field in Table 12
- **`SrmCertificateChain`** – *SRM Certificate Chain* field in Table 12
- **`EncryptedAuthRespData`** – Encrypted **`AuthRespData`** field in Table 12

6.2.1.3 Exception Handling

There may be an unexpected exception during the Authentication Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response, finds an error by referring to the *Status*, fails to verify the *SRM Certificate Chain*, or fails to decrypt the *Encrypted AuthResp Data*, then the DRM Agent regards it as an exception and terminates communication by discarding any existing SAC context with the SRM Agent. The User may be informed of this exception.

6.2.2 Key Exchange

This step performs the key exchange and key confirmation.

6.2.2.1 Description of Messages

The DRM Agent generates a random number (RN_D), and encrypts it with the SRM's public key. At this step, the DRM Agent also encrypts the hash of the SRM Random Number (RN_S) received in the AuthenticationResponse.

Then the DRM Agent sends the KeyExchangeRequest to exchange keys with the SRM Agent. The fields of the request are defined in Table 14.

Table 14: Fields of KeyExchangeRequest

Fields	Protection Requirement	Description
Encrypted KeyEx Data	No	E ($PuKey_S$, KeyExData) where KeyExData = RN_D | $H(RN_S)$ | *Selected Version* *Selected Version* is identical to the *Selected Version* received by the DRM Agent in the SrmHelloResponse. *KeyExData* is encrypted with the SRM's public key ($PuKey_S$) under the trust anchor sent in the AuthenticationRequest.

Upon receiving the KeyExchangeRequest, the SRM Agent decrypts *Encrypted KeyExData* with the SRM's private key (under the trust anchor sent in the AuthenticationRequest).

The SRM Agent compares the decrypted $H(RN_S)$ to the hash of the random number (RN_S) that the SRM Agent sent in the AuthenticationResponse. The hash is computed using the negotiated algorithm. The SRM Agent also compares the decrypted *Selected Version* to the *Selected Version* field sent in the SrmHelloResponse.

After this action, the SRM Agent sends the KeyExchangeResponse to carry the result of the action. The fields of the response are defined in Table 15**.**

Table 15: Fields of KeyExchangeResponse

Fields	Protection Requirement	Description
Status	No	The result of processing the KeyExchangeRequest message. The *Status* values are specified in Table 16. If *Status* contains any error, only this field is present in the KeyExchangeResponse.
Hash Of RanNum Data	No	H(RanNumData) where RanNumData = RN_D | RN_S. RanNumData is hashed by the selected hash algorithm.

Table 16: Status of Key Exchange Message

Status Value	Description
Success	The request was successfully processed.
Field Decryption Failed	The SRM Agent fails to decrypt the encrypted fields.
SRM Random Number Mismatched	The SRM Random Number from the DRM Agent is not identical to its original value in the SRM.
Version Mismatched	The *Selected Version* received in KeyExchagenRequest is not matched with the original value sent in the SrmHelloResponse.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the KeyExchangeResponse and *Status* is *Success*, the DRM Agent confirms whether the hash of the concatenation of the Device Random Number (RN_D) and the SRM Random Number (RN_S) matches the corresponding hash of the random numbers exchanged in the KeyExchangeRequest and AuthenticationResponse respectively.

After the key exchange and key confirmation are successfully finished, the DRM Agent and SRM Agent generate security elements by using the Key Derivation Function as specified in section 6.3.1.

6.2.2.2 Format of Messages

The message format (**`MessageBody`**) of the KeyExchangeRequest is specified as follows. The **`messageType`** is set to '0' and the message is not protected by an HMAC.

```
DeviceRandomNumber() {
    RandomNumber()                  // Defined in Appendix E.1
}

SrmRandomNumber() {
    RandomNumber()                  // Defined in Appendix E.1
}

HashOfSrmRandomNumber() {
    Hash()                          // Defined in Appendix E.1
}

SelectedVersion() {
    Version()                       // Defined in Appendix E.2.1
}
```

```
KeyExData() {
    DeviceRandomNumber()
    HashOfSrmRandomNumber()
    SelectedVersion()
}


EncryptedKeyExData() {
    EncryptedData()                // Defined in Appendix E.1
}


MessageBody() {
    EncryptedKeyExData()
}
```

The fields are defined as follows:

- **KeyExData** – *KeyExData* value of *Encrypted KeyEx Data* field in Table 14
- **DeviceRandomNumber** – RN_D value of *Encrypted KeyEx Data* field in Table 14
- **SrmRandomNumber** – RN_S value of *Encrypted KeyEx Data* field in Table 14
- **Selected Version** – *Selected Version* value of *Encrypted KeyEx Data* field in Table 14
- **HashOfSrmRandomNumber** – Hash of the **SrmRandomNumber** using the selected hash algorithm
- **EncryptedKeyExData** – **KeyExData** encrypted with the SRM's public key

The message format (**MessageBody**) of the KeyExchangeResponse is specified as follows. The **messageType** is set to '1' and the message is not protected by an HMAC.

```
RanNumData() {
    DeviceRandomNumber()
    SrmRandomNumber()
}


HashOfRanNumData() {
    Hash()                         // Defined in Appendix E.1
}


MessageBody() {
    Status()                       // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        HashOfRanNumData()
    }
}
```

The fields are defined as follows:

- **RanNumData** – *RanNumData* value of *Hash Of RanNum Data* field in Table 15
- **DeviceRandomNumber** – RN_D value of *Hash Of RanNum Data* field inTable 15
- **SrmRandomNumber** – RN_S value of *Hash Of RanNum Data* field in Table 15
- **Status** - *Status* field in Table 15
- **HashOfRanNumData** – Hash of **RanNumData** field in Table 15 using the selected hash algorithm

6.2.2.3 Exception Handling

There may be an unexpected exception during the Key Exchange Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response, finds an error by referring to the *Status*, or fails to verify the random numbers, then the DRM Agent regards it as an exception and terminates communication by discarding any existing SAC context with the SRM Agent. The User may be informed of this exception.

6.3 Secure Authenticated Channel

Whenever sensitive information, such as cryptographic keys, needs to be transferred between the DRM Agent and SRM Agent, a Secure Authenticated Channel (SAC) needs to be used. A SAC is a logical channel that provides message integrity and optionally message confidentiality. A SAC needs to be established using credentials from a trust model under which the sensitive information was created.

In the SRM 1.0, if the DRM Agent and SRM Agent support more than one trust model in common, then multiple SACs can be established by repeating the MAKE process.

In the SRM 1.1, if the DRM Agent and SRM Agent support more than one trust model in common, then an existing SAC SHALL be replaced by a SAC using another supported trust model by executing a new MAKE process.

6.3.1 Key Derivation Function

After the MAKE process is completed, both the DRM Agent and the SRM Agent have mutually authenticated each other and have exchanged secret random numbers that are used in generating key materials (Session Key and MAC Key). The keys are used in the SAC.

The Key Derivation Function (KDF) is the same as the KDF specified in section 7.1.2 of the OMA DRM v2.0 specification [OMADRMv2.0]. A trust model may use a different KDF. The following key material in Table 17 is derived from the KDF. When using the KDF, let **Z** = RN_D | RN_S (RN_D and RN_S include the length fields.), **otherInfo** = *Supported Algorithms* | *Selected Algorithms*, and **kLen** is 36 bytes (the total size of the key materials in Table 17).

Table 17: Key Materials

Fields	Size	Description	Nomenclature
MAC Key	160 bits	HMAC-SHA1 Key: The first 20 octets of ***T*** as the derived key	MK_0
Session Key	128 bits	AES Key: The next 16 octets of ***T*** as the derived key	SK

By default, the DRM Agent and SRM Agent support the AES128-CBC mode. The padding is performed as specified in [RFC2630].

The formats of RN_D and RN_S are specified in section 6.2.2.2 (**DeviceRandomNumber** and **SrmRandomNumber**) and the formats of *Supported Algorithms* and *Selected Algorithms* are specified in section 6.2.1.2 (**SupportedAlgorithms** and **SelectedAlgorithms**).

6.3.2 SAC Context

Once a SAC has been established, a logical SAC context will exist. The context consists of the following information:

- MAC Key – this key gets updated as specified in section 6.3.4.
- Session Key – this key does not change for the duration of the SAC.
- Selected Algorithms – the algorithms that were negotiated during the MAKE process.
- Trust Anchor – the trust anchor under which the SAC was established. Used when multiple SACs are available and the Device wants to switch to a different SAC as specified in section 6.3.5

- Entity ID – for the Device, this contains the SRM's ID (under the trust anchor); for the SRM, this contains the Device's ID (under the trust anchor).

The SAC context exists until a new SAC with the same Device and SRM, under the same trust model, is established. By using the SRM Hello message pair, a DRM Agent can determine if it communicating with the same SRM. If the DRM Agent reuses the SAC context, sends a secure message and gets back a *Field Integrity Verification Failed* error, this probably indicates that the SAC context is no longer valid. The DRM Agent SHOULD establish a new SAC.

6.3.3 Secure Message

Once the SAC has been established, two types of security are provided. The first type is integrity protection and the other type is confidentiality protection. The integrity protection is performed by generating HMAC (using the negotiated HMAC algorithm) over fields using the current MAC Key (MK) in the SAC Context. The confidentiality protection is performed by encrypting fields using the current Session Key (SK) and the negotiated symmetric encryption algorithm.

6.3.4 Message Replay Protection

Replay protection is provided by using a different MAC Key for every request or response message that requires integrity protection. After the SRM Agent sends a response, if the request or the response required integrity protection, then the SRM Agent MUST generate a new MK. After the DRM Agent receives a response, if the request or the response required integrity protection, the DRM Agent MUST generate a new MK before sending a request needing integrity protection. Using the SAC Context, a new MK is generated as follows:

$MK_{i+1} = H(MK_i)$, where H is the negotiated hashing algorithm

6.3.5 Changing SAC

The DRM Agent changes to a different SAC as illustrated in the following Figure 6:

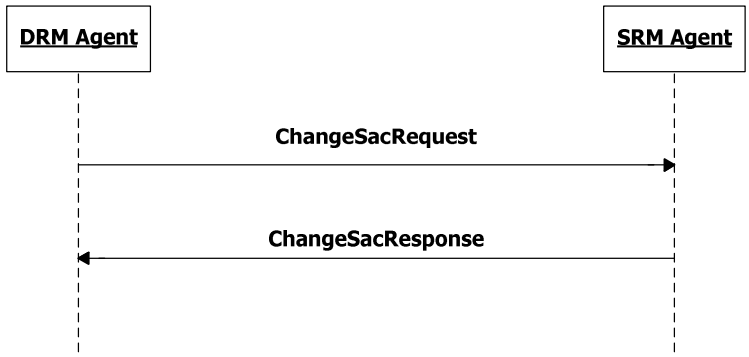

Figure 6: Sequence Diagram – Change SAC

If an SRM supports multiple trust models, then the SRM MAY implement the Change SAC message pair. If this message pair is supported, this is indicated in the SrmHelloResponse (see section 6.1). If this message pair is not supported, then the DRM Agent MUST use the MAKE process to change to a different SAC. Note that the DRM Agent can use the SRM Hello message pair to determine if it is communicating with the same SRM.

6.3.5.1 Description of Messages

If the SRM supports multiple trust models and the DRM Agent has established multiple SACs (as described in section 6.3), the DRM Agent can send the ChangeSacRequest so that the SRM Agent can change to a different SAC. The fields of the request are defined in Table 18.

Table 18: Fields of ChangeSacRequest

Field	Protection Requirement	Description
Trust Anchor	No	The Trust Anchor under which a SAC was established.

Upon receiving the ChangeSacRequest, the SRM Agent checks that it has established a SAC under the specified trust anchor.

The SRM Agent sends the ChangeSacResponse to carry the result of the processing the request. Upon sending the ChangeSacResponse and if the *Status* is *Success*, the SRM Agent MUST change to the SAC identified by the trust anchor and start using that SAC context. The fields of the response are defined in Table 19.

Table 19: Fields of ChangeSacResponse

Field	Protection Requirement	Description
Status	No	The result of processing the ChangeSacRequest message. The *Status* values are specified in Table 20.

Table 20: Status of Change SAC Message

Status Value	Description
Success	The request was successfully handled.
SAC Not Established	A SAC under the trust anchor has not been established.
Request Not Supported	The SRM only supports one trust model and hence does not support this request.
Unknown Error	Other errors

Upon receiving the ChangeSacResponse and *Status* is *Success*, the DRM Agent MUST change to the SAC and start using that SAC context.

6.3.5.2 Format of Messages

The message format (**`MessageBody`**) of the ChangeSacRequest is specified as follows. The **`messageType`** is set to '0' and the message is not protected by an HMAC.

```
MessageBody() {
    TrustAnchor()                  // Defined in section 6.1.1.2
}
```

The fields are defined as follows:

- **`TrustAnchor`** – the Trust Anchor field in Table 18.

The message format (**`MessageBody`**) of the ChangeSacResponse is specified as follows. The **`messageType`** is set to '1' and the message is not protected by an HMAC.

```
MessageBody() {
    Status()
}
```

The fields are defined as follows:

- **`Status`** – Status field in Table 19.

6.3.5.3 Exception Handling

There may be an unexpected exception during the Change SAC message pair processing as specified in section 5.5.1. If the DRM Agent fails to receive the response or finds an *Unknown Error* by referring to the *Status*, then the DRM Agent regards it as an exception and terminates communication by discarding any existing SAC context with the SRM Agent. The User may be informed of this exception.

6.3.6 Maintenance of Multiple SACs

For "S2S Rights Move" transaction, the two SACs between DRM Agent and two SRMs SHALL be set up successfully before DRM Agent initiates the "S2S Rights Move" while during the transaction changing SAC is normally conducted when it is needed.

6.4 Revocation Status Checking

Revocation status checking between the SRM Agent and the DRM Agent is a necessary procedure that MUST occur before exchanging any message over the SAC. During mutual authentication between the DRM Agent and SRM Agent, revocation status checking is performed locally by using a cached Certificate Revocation List (CRL). A DRM Agent MUST cache a CRL that contains revocation status about SRMs, and the SRM Agent MUST cache a CRL that contains revocation status about Devices. If the connected SRM or Device, respectively, is on the CRL then the SAC MUST be terminated. Furthermore, the SAC MUST be terminated if the SRM has information that the DRM Agent has been revoked (see section 6.4.3). The validity dates for the cached CRL (whether in the DRM Agent or the SRM Agent) does not need to be checked for revocation status checking. The CRL update schedule and CRL distribution and thereby criteria for ensuring valid CRLs are beyond the scope of this specification. This section specifies protocols that relevant trust models may require to implement revocation checking.

Note: This Enabler does not require revocation status checking of the RI certificate chain when verifying RI signatures during Move or Local Rights Consumption. However, Devices MUST follow [OMADRMv2.1] requirements when performing ROAP.

For a DRM Agent and an SRM Agent to update an old CRL with a newer CRL, this document specifies protocols for the following purposes:

- CRL Information Exchange (Refer to section 6.4.1)

  The DRM Agent and SRM Agent exchange CRL numbers in order to determine if CRL(s) in the Device supersede CRL(s) in the SRM or vice versa.

- OCSP Nonce (Refer to section 6.4.2)

  The DRM Agent requests a nonce from the SRM Agent. The DRM Agent uses the nonce for the OCSP request so that the SRM Agent can be provided with the current DRM time and check the revocation status of the Device.

- OCSP Response Processing (Refer to section 6.4.3)

  The DRM Agent passes an OCSP response to the SRM Agent that includes the revocation status of the DRM Agent and the DRM time.

- CRL Delivery from Device to SRM (Refer to section 6.4.4)

  The DRM Agent sends its CRL(s) to the SRM Agent. The SRM Agent replaces the CRL(s) stored in the SRM with the received CRL(s).

- CRL Delivery from SRM to Device (Refer to section 6.4.5)

  The DRM Agent retrieves CRL(s) in SRM, and replaces its stored CRL(s) with the retrieved CRL(s).

Some trust models may use an OCSP responder to provide the revocation status of Devices. To enable the SRM Agent to use OCSP to check the revocation status of Devices, the DRM Agent SHOULD support the following:

- OCSP communication protocol between an OCSP responder and the DRM Agent as specified in [OCSP-MP]
- OCSP Nonce request in section 6.4.2
- OCSP request generation with the nonce provided by the SRM Agent
- OCSP Response Processing between the DRM Agent and SRM Agent in section 6.4.3

In order to use OCSP to check the revocation status of Devices, the SRM Agent SHOULD also support the following:

- OCSP Nonce request in section 6.4.2
- OCSP Response Processing in section 6.4.3

The CRL(s) are updated by the following procedure.

The CRL information exchange function in section 6.4.1 is executed.

If the DRM Agent supports the OCSP responder – DRM Agent communication, then the DRM Agent MAY pass the OCSP response to the SRM Agent by using the OCSP Nonce request function in section 6.4.2 and OCSP response processing function in section 6.4.3.

If the CRL information exchange function finds that CRL(s) must be updated, then the CRL delivery function in section 6.4.4 or 6.4.5 is used.

To minimize the impact of not checking the CRL validity dates, the following procedure is introduced

- Event Counting with a threshold as specified in Appendix J

  The DRM Agent and the SRM Agent count events until a predefined threshold is reached, upon which a "fresh" CRL is required. Support for Event Counting is OPTIONAL; relevant trust models may mandate the use of the event counting mechanism.

6.4.1 CRL Information Exchange

The DRM Agent reads an SRM's CRL information list in order to determine if CRL(s) in the Device supersede CRL(s) in the SRM or if CRL(s) in the SRM supersede CRL(s) in the Device as illustrated in Figure 7.

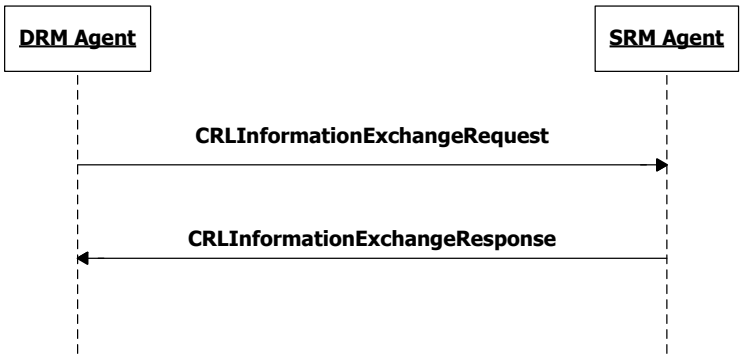

Figure 7: Sequence Diagram – CRL Information Exchange

6.4.1.1 Description of Messages

The DRM Agent sends the CRLInformationExchangeRequest to read the CRL Information List from the SRM. The CRLInformationExchangeRequest has no fields.

Upon receiving the CRLInformationExchangeRequest, the SRM Agent sends the CRLInformationExchangeResponse to the DRM Agent. The fields of the response are defined in Table 21.

Table 21: Fields of CRLInformationExchangeResponse

Fields	Protection Requirement	Description
Status	No	The result of processing the CRLInformationExchangeRequest message. The *Status* values are specified in Table 22. If *Status* contains any error, only this field is present in the CRLInformationExchangeResponse.
CRL Information List	No	CRL Information is a pair of CRL Issuer ID and CRL Number. The *CRL Information List* contains CRL Information of all CRLs in the SRM. CRL Issuer ID is the 160-bit SHA-1 hash of the public key corresponding to the private key used to sign the CRL (i.e. the keyIdentifier field of the authorityKeyIdentifier component in the CRL). The CRL Number is the value contained in the CRL number extension of the referenced CRL. This value is used to determine when a particular CRL supersedes another CRL.

Table 22: Status of CRL Information Exchange Message

Status Value	Description
Success	The request was successfully processed.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	The SRM Agent fails to read the CRL Information.

Upon receiving the CRLInformationExchangeResponse, the DRM Agent checks the *Status* field. If *Status* is *Success*, the DRM Agent compares each element in the *CRL Information List* received from the SRM with the Device's list. If the CRL Issuer IDs are identical, then the CRL numbers are compared to determine if the SRM's CRL supersedes the CRL stored in the Device or if the Device's CRL supersedes the CRL stored in the SRM.

If the DRM Agent finds that CRL(s) in the Device supersedes CRL(s) in the SRM from the same CRL issuer, the DRM Agent MUST transfer the new CRL(s) to the SRM using the CRLUpdateRequest as specified in section 6.4.4. If there are multiple CRL(s) to be updated, the DRM Agent repeats the CRL update request.

If the DRM Agent finds that CRL(s) in the SRM supersede CRL(s) in the Device from the same CRL issuer, the DRM Agent MUST retrieve the new CRL(s) from the SRM Agent using the CRLRetrievalRequest as specified in section 6.4.5. If there are multiple CRL(s) to be updated, the DRM Agent repeats the CRL retrieval request.

6.4.1.2 Format of Messages

The message format (**`MessageBody`**) of the CRLInformationExchangeRequest is empty. The **`messageType`** is set to'0' and the message is not protected by an HMAC.

The message format (**`MessageBody`**) of the CRLInformationExchangeResponse is specified as follows. The **`messageType`** is set to '1' and the message is not protected by an HMAC.

```
CrlIssuerId() {
    OctetString8()                  // Defined in Appendix E.1
}


CrlNumber() {
    OctetString8()                  // Defined in Appendix E.1
}


CrlInformation() {
    CrlIssuerId()
    CrlNumber()
}


CrlInformationList() {
    nbrOfCrlInformation      8      uimsbf
    for ( i = 0 ; i < nbrOfCrlInformation ; i++ ) {
        CrlInformation()
    }
}


MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        CrlInformationList()
    }
}
```

The fields are defined as follows:

- **`Status`** - *Status* field in Table 21
- **`CrlInformationList`** – *CRL Information List* field in Table 21

6.4.1.3 Exception Handling

There may be an unexpected exception during the CRL Information Exchange Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response or finds an error by referring to the *Status*, then the DRM Agent regards it as an exception and terminates communication by discarding any existing SAC context with the SRM Agent. The User may be informed of this exception.

6.4.2 OCSP Nonce

This section is valid only for DRM Agents that support communications with an OCSP Responder. If supported by the SRM Agent, a DRM Agent can get a nonce, for use in an OCSP Request, from an SRM Agent as illustrated in Figure 8.

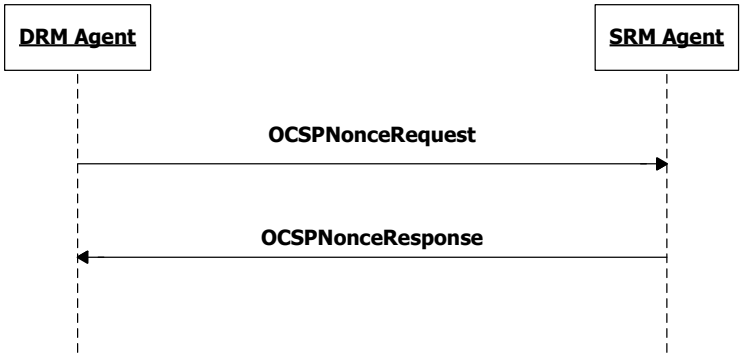

Figure 8: Sequence Diagram – OCSP Nonce

6.4.2.1 Description of Messages

The DRM Agent sends the OCSPNonceRequest to request the SRM Agent to generate a nonce. There are no fields included in this request.

Upon receiving the OCSPNonceRequest, the SRM Agent generates a nonce (i.e. OCSP Nonce) and returns the value by sending the OCSPNonceResponse to the DRM Agent.

If the SRM Agent does not support the OCSP response processing, it MUST return the error code - *Request Not Supported*.

The fields of the response are defined in Table 23.

Table 23: Fields of OCSPNonceResponse

Fields	Protection Requirement	Description
Status	No	The result of processing the OCSPNonceRequest message. The Status values are specified in Table 24. If *Status* contains any error, only this field is present in the OCSPNonceResponse.
OCSP Nonce	No	This is a number randomly generated by the SRM Agent.
OCSP Responder Key Identifier	No	This field identifies a trusted OCSP responder key stored in the SRM. If the identifier matches the key in the certificate used by the Device's OCSP responder, the DRM Agent MAY remove the OCSP Responder certificate chain from the OCSP response before providing the OCSP response to the SRM.

Table 24: Status of OCSP Nonce Message

Status Value	Description
Success	The request was successfully processed.
Request Not Supported	The SRM Agent does not support the OCSP response processing.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	The SRM Agent fails to generate the *OCSP Nonce*.

Upon receiving the response (*Status* is *Success*), the DRM Agent generates a nonce based OCSP request for its own certificate (using the OCSP nonce provided by the SRM Agent) and sends it to the OCSP responder. The *OCSP Nonce* is identified by the object identifier *id-pkix-ocsp-nonce*, while the *extnValue* is the value of the nonce.

6.4.2.2 Format of Messages

The message format (**MessageBody**) of the OCSPNonceRequest is specified as follows. The **messageType** is set to'0' and the message is not protected by an HMAC. This message does not include fields.

The message format (**MessageBody**) of the OCSPNonceResponse is specified as follows. The **messageType** is set to '1' and the message is not protected by an HMAC.

```
OcspNonce() {
    // The length of the nonce MUST be
    // at least 14 bytes and no more than 32 bytes
    OctetString8()                  // Defined in Appendix E.1
}


OcspResponderKeyIdentifier() {
    OctetString8()                  // Defined in Appendix E.1
}


MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        OcspNonce()
        OcspResponderKeyIdentifier()
    }
}
```

The field is defined as follows:

- **Status** - *Status* field in Table 23
- **OcspNonce** – *OCSP Nonce* field in Table 23
- **OcspResponderKeyIdentifier** – *OCSP Responder Key Identifier* field in Table 23

6.4.2.3 Exception Handling

There may be unexpected exceptions during the OCSP Nonce Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response or receives an error in the *Status* field (other than *Request Not Supported*), then the DRM Agent regards it as an exception and terminates communication by discarding any existing SAC context with the SRM Agent. The User may be informed of this exception.

6.4.3 OCSP Response Processing

This section is valid only for DRM Agents that support communications with an OCSP responder. If supported by the SRM Agent, the DRM Agent sends an OCSP Response to the SRM Agent as illustrated in Figure 9.

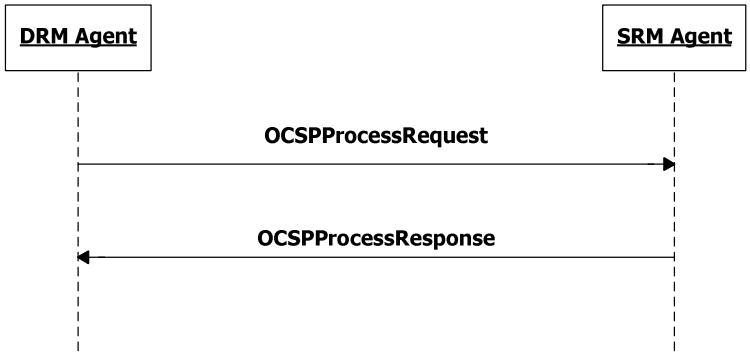

Figure 9: Sequence Diagram – OCSP Processing

6.4.3.1 Description of Messages

Upon receiving the OCSP response from the OCSP responder, the DRM Agent sends the OCSPProcessRequest to pass the response to the SRM Agent. The fields of the request are defined in Table 25.

Table 25: Fields of OCSPProcessRequest

Fields	Protection Requirement	Description
OCSP Response	No	*OCSP Response* contains the revocation status of the DRM Agent, the time at which the OCSP Response was signed, and the *OCSP Nonce* transferred by the OCSPNonceResponse as specified in section 6.4.2.

Upon receiving the OCSPProcessRequest, the SRM Agent verifies the *OCSP Response*. The SRM Agent MUST verify that the OCSP-provided status of all revocable certificates in the Device Certificate Chain is *good* (refer to [OCSP-MP]). If the status is *revoked*, the SRM Agent SHOULD keep a record of that status for future use (refer to section 6.4). The SRM Agent MUST be able to detect that an OCSP responder certificate is non-revocable through the use of the *id-pkix-ocsp-nocheck* extension as specified in [OMADRMv2.0]. The determination of which certificates in a Device Certificate Chain are revocable is deemed to be part of the trust model of the root of trust of that chain. In case the root of trust does not specify such a policy, the SRM SHALL assume a default model. In the default model only the Device Certificate is revocable and requires an OCSP response to prove its status.

SRM Agents MUST be able to match a nonce sent for OCSP purposes in the OCSPNonceResponse (in section 6.4.2) with a nonce in the received OCSP Response.

With the OCSP response, the SRM Agent is able to verify the revocation status of the Device Certificate during the MAKE process in section 6.2 and can check the freshness of CRL(s) based on the *producedAt* time in the OCSP response.

The SRM Agent then sends the OCSPProcessResponse. The fields of the response are defined in Table 26. If the SRM Agent does not support the OCSP response processing, it MUST return the error code - *Request Not Supported*.

Table 26: Fields of OCSPProcessResponse

Fields	Protection Requirement	Description
Status	No	The result of processing the OCSPProcessRequest message. The *Status* values are specified in Table 27.

Table 27: Status of OCSP Process Message

Status Value	Description
Success	The request was successfully processed.
OCSP Response Verification Failed	The SRM Agent fails to verify the *OCSP Response*.
Invalid OCSP Nonce	The OCSP nonce in the OCSP response is not identical with the OCSP nonce generated by the SRM Agent.
Request Not Supported	The SRM Agent does not support the OCSP response processing.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

If no errors or exceptions (*Status* = *Success* or *Request Not Supported*), the OCSP Process Message processing is completed.

6.4.3.2 Format of Messages

The message format (**`MessageBody`**) of the OCSPProcessRequest is specified as follows. The **`messageType`** is set to'0' and the message is not protected by an HMAC.

```
OcspResponse() {
    OctetString16()                // Defined in Appendix E.1
}

MessageBody() {
    OcspResponse()
}
```

The field is defined as follows:

- **`OcspResponse`** – *OCSP Response* field in Table 25

The message format (**`MessageBody`**) of the OCSPProcessResponse is specified as follows. The **`messageType`** is set to '1' and the message is not protected by an HMAC.

```
MessageBody() {
    Status()                       // Defined in Appendix E.2.2
}
```

The field is defined as follows:

- **`Status`** - *Status* field in Table 26

6.4.3.3 Exception Handling

There may be an unexpected exception during the OCSP Process Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response, or finds an error by referring to the *Status* (except *Request Not Supported*), then the DRM Agent regards it as an exception and terminates communication by discarding any existing SAC context with the SRM Agent. The User may be informed of this exception.

6.4.4 CRL Delivery from Device to SRM

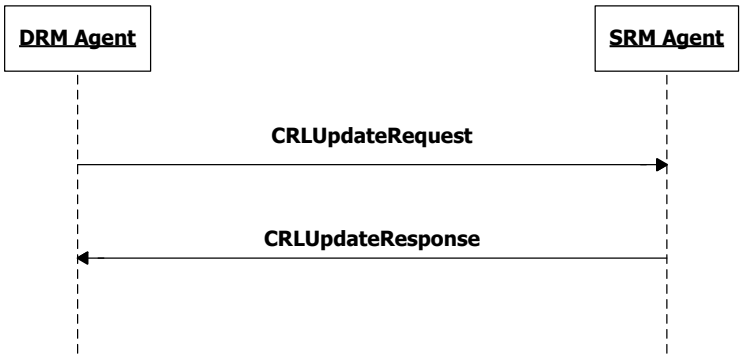

Figure 10: Sequence Diagram – CRL Delivery from Device to SRM

6.4.4.1 Description of Messages

The DRM Agent sends the CRLUpdateRequest to replace the current CRL in the SRM with the CRL in the Device. The fields of the request are defined in Table 28.

Table 28: Fields of CRLUpdateRequest

Fields	Protection Requirement	Description
CRL	No	Certificate Revocation List (CRL) contains revocation status information for Device Certificates and/or SRM Certificates. Refer to Appendix H.2

Upon receiving the CRLUpdateRequest, the SRM Agent verifies the signature over the *CRL*. If the signature of the *CRL* is valid, and the received *CRL* is newer than the CRL of the SRM, then the SRM Agent replaces the current CRL in the SRM with the received *CRL*. The CRL numbers are compared to determine if the received *CRL* supersedes the CRL stored in the SRM.

The SRM Agent can recognize the CRL issuer by referring to the **authorityKeyIdentifier** component in the *CRL*.

The SRM Agent then sends the CRLUpdateResponse. The fields of the response are defined in Table 29.

Table 29: Fields of CRLUpdateResponse

Fields	Protection Requirement	Description
Status	No	The result of processing the CRLUpdateRequest message. The *Status* values are specified in Table 30.

Table 30: Status of CRL Update Message

Status Value	Description
Success	The request was successfully processed.
Old CRL	*CRL* in the request is older than the CRL in SRM.
CRL Verification Failed	The verification of the signature over *CRL* is failed.
Trust Anchor Not Supported	The issuer of the CRL in the request is not supported by the SRM Agent.
Parameter Failed	A field in the request has an invalid length or structure.

Status Value	Description
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

6.4.4.2 Format of Messages

The message format (**MessageBody**) of the CRLUpdateRequest is specified as follows. The **messageType** is set to '0' and the message is not protected by an HMAC.

```
MessageBody() {
    Crl()                          // Defined in Appendix E.1
}
```

The field is defined as follows:

- **Crl** - *CRL* field in Table 28

The message format (**MessageBody**) of the CRLUpdateResponse is specified as follows. The **messageType** is set to '1' and the message is not protected by an HMAC.

```
MessageBody() {
    Status()                       // Defined in Appendix E.2.2
}
```

The field is defined as follows:

- **Status** - *Status* field in Table 29

6.4.4.3 Exception Handling

There may be an unexpected exception during the CRL Update Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response or finds an error by referring to the *Status*, then the DRM Agent regards it as an exception and terminates communication by discarding any existing SAC context with the SRM Agent. The User may be informed of this exception.

6.4.5 CRL Delivery from SRM to Device

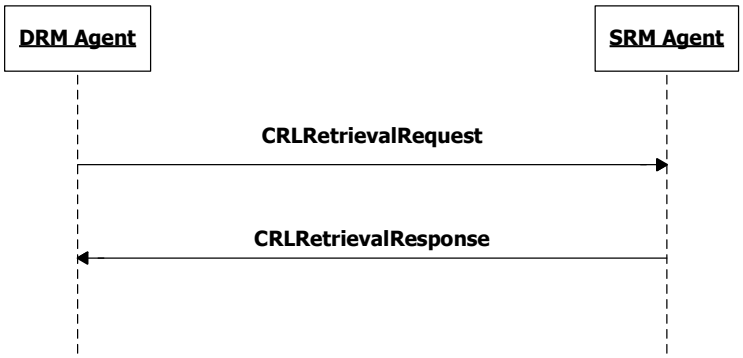

Figure 11: Sequence Diagram – CRL Delivery from SRM to Device

6.4.5.1 Description of Messages

The DRM Agent sends the CRLRetrievalRequest to retrieve the CRL in the SRM. The fields of the request are defined in Table 31.

Table 31: Fields of CRLRetrievalRequest

Fields	Protection Requirement	Description
CRL Issuer ID	No	The 160-bit SHA-1 hash of the public key corresponding to the private key used to sign the CRL (i.e. the keyIdentifier field of the authorityKeyIdentifier component in the CRL).

Upon receiving the CRLRetrievalRequest, the SRM Agent retrieves the CRL stored in the SRM that corresponds to the *CRL Issuer ID*.

The SRM Agent then sends the CRLRetrievalResponse to carry the result of the action. The fields of the response are defined in Table 32.

Table 32: Fields of CRLRetrievalResponse

Fields	Protection Requirement	Description
Status	No	The result of processing the CRLRetrievalRequest message. The *Status* values are specified in Table 33. If *Status* contains any error, only this field is present in the CRLRetrievalResponse.
CRL	No	Certificate Revocation List (CRL) contains revocation status information for Device Certificates and/or SRM Certificates. Refer to Appendix H.2

Table 33: Status of CRL Retrieval Message

Status Value	Description
Success	The request was successfully processed.
CRL Not Found	There is no CRL corresponding to the *CRL Issuer ID*.

Status Value	Description
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the CRLRetrievalResponse, the DRM Agent verifies the signature over the *CRL*. If the signature of the *CRL* is valid, and the retrieved *CRL* is newer than the CRL of the Device, then the DRM Agent replaces the current CRL in the Device with the retrieved *CRL*. The CRL numbers are compared to determine if the received *CRL* supersedes the CRL stored in the Device.

6.4.5.2 Format of Messages

The message format (**`MessageBody`**) of the CRLRetrievalRequest is specified as follows. The **`messageType`** is set to '0' and the message is not protected by an HMAC.

```
CrlIssuerId() {
    OctetString8()                  // Defined in Appendix E.1
}

MessageBody() {
    CrlIssuerId()
}
```

The field is defined as follows:

- **`CrlIssuerId`** – *CRL Issuer ID* field in Table 31

The message format (**`MessageBody`**) of the CRLRetrievalResponse is specified as follows. The **`messageType`** is set to '1' and the message is not protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        Crl()                       // Defined in Appendix E.1
    }
}
```

The fields are defined as follows:

- **`Status`** - *Status* field in Table 32
- **`Crl`** - *CRL* field in Table 32

6.4.5.3 Exception Handling

There may be an unexpected exception during the CRL Retrieval Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response, finds an error by referring to the *Status* or fails the CRL verification, then the DRM Agent regards it as an exception and terminates communication by discarding any existing SAC context with the SRM Agent. The User may be informed of this exception.

6.5 Movement of Rights from Device to SRM

Rights are Moved from a Device to an SRM as illustrated in Figure 12. As shown in Figure 12, this transaction is comprised of two request/response message pairs: Installation Setup message pair and Rights Installation message pair. The Installation Setup message pair SHOULD be followed by the Rights Installation message pair. If an SRM Agent receives any request message other than the RightsInstallationRequest following receipt of the InstallationSetupRequest, the SRM Agent SHOULD return *Unexpected Request* in the *Status* field of the response message.

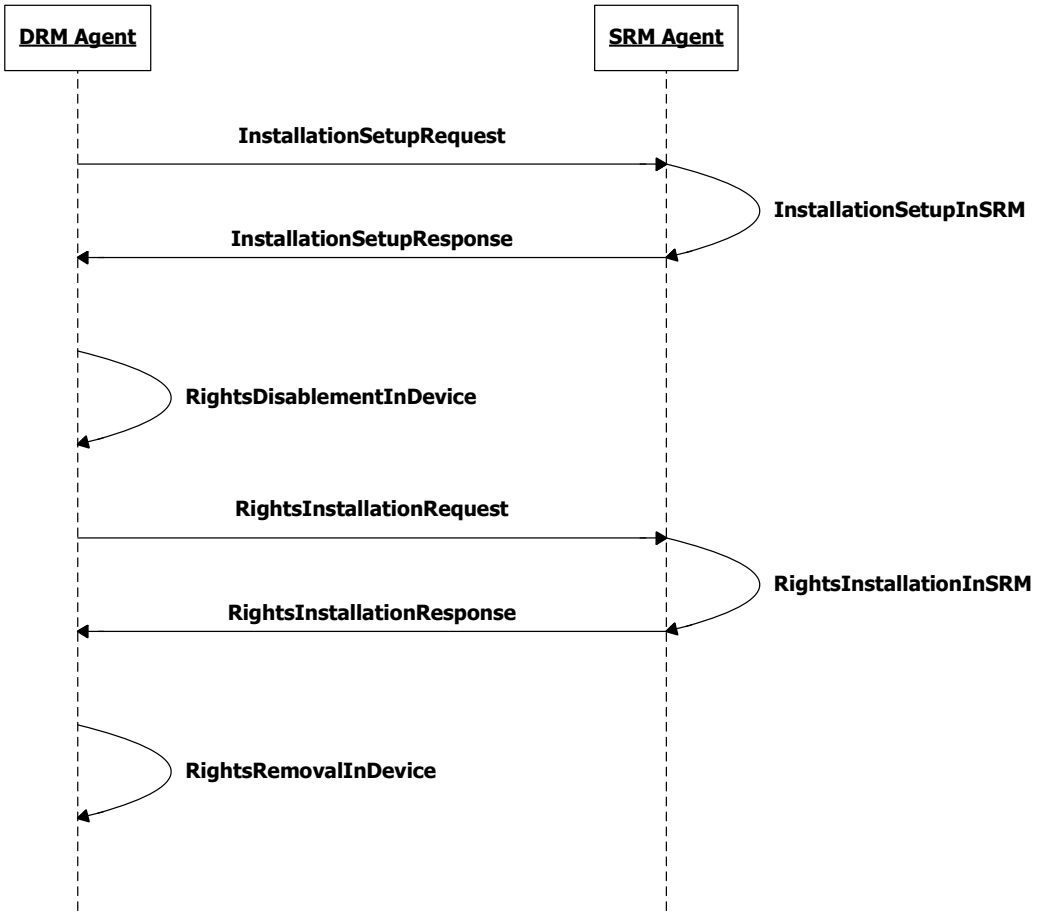

Figure 12: Sequence Diagram – Movement of Rights from Device to SRM

Before sending the InstallationSetupRequest message, the DRM Agent MUST check the following in the Rights Object that will be Moved:

1) Check if the Rights Object has the <move> permission. If there is no <move> permission, then do not perform the Move transaction.
2) Check if the <move> permission has a <system> constraint. If there is no <system> constraint, proceed with sending the InstallationSetupRequest; else check the <context> child element(s) of the <system> constraint. If any <context> child element identifies the SRM protocol, then proceed with sending the InstallationSetupRequest; else do not perform the Move transaction.

Before sending the RightsInstallationRequest message, the DRM Agent MUST check the following in the Rights Object that will be Moved:

1) Check if the <move> permission has a <count> constraint. If there is a <count> constraint, then check the current count value in the state information. If the current count is 0, then do not perform the Move transaction. Otherwise (current count > 0), decrement current count value.

6.5.1 Installation Setup

6.5.1.1 Description of Messages

The DRM Agent sends the InstallationSetupRequest to initiate a Move to the SRM. The fields of the request are defined in Table 34.

Table 34: Fields of InstallationSetupRequest

Fields	Protection Requirement	Description
Handle	Integrity & Confidentiality	The *Handle* identifies the Rights while stored in the SRM. It is a 10 byte random value generated by the DRM Agent for this Move transaction. Refer to section 5.1.3.
Size of Rights	Integrity	Size of Rights in bytes. This informs the SRM Agent the size of Rights that will be installed in the SRM as specified in section 6.5.3. *Size of Rights* = Length of **`RightsInformation`**. **`RightsInformation`** is specified in section 6.5.3.2.

Upon receiving the InstallationSetupRequest, the SRM Agent MUST perform the following procedure:

1. Verify the integrity of the fields
2. Decrypt the *Handle* with the Session Key
3. The SRM Agent MUST check if the SRM already has the same Handle. If yes, the SRM Agent sets *Status* to *Duplicate Handle* and the SRM Agent sends the InstallationSetupResponse as described below.
4. The SRM Agent checks if the SRM has space for the new Rights. If not, the SRM Agent sets *Status* to *Not Enough Space*. Otherwise, the SRM Agent stores the *Handle* in the SRM securely. The *Handle* is not included in the Handle List until the Move transaction is completed.

The SRM Agent sends the InstallationSetupResponse to carry the result of the procedure. The fields of the response are defined in Table 35.

Table 35: Fields of InstallationSetupResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the InstallationSetupRequest message. The *Status* values are specified in Table 36.

Table 36: Status of Installation Setup Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Duplicate Handle	The SRM already has the Handle and its corresponding Rights.
Not Enough Space	The SRM does not have enough space to store Rights having the same size as the *Size of Rights*.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.

Status Value	Description
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response.

If no errors or exceptions (*Status* = *Success*), the DRM Agent continues with section 6.5.2.

6.5.1.2 Format of Messages

The message format (**`MessageBody`**) of the InstallationSetupRequest is specified as follows. The **`messageType`** is set to ’0’ and the message is protected by an HMAC.

```
MessageBody() {
    EncryptedHandle()               // Defined in Appendix E.2.8
    sizeOfRights            16      uimsbf
}
```

The fields are defined as follows:

- **`Handle`** –*Handle* field in Table 34
- **`sizeOfRights`** –*Size Of Rights* field in Table 34
- **`EncryptedHandle`** – Encrypted **`Handle`** with the current Session Key (SK)

The message format (**`MessageBody`**) of the InstallationSetupResponse is specified as follows. The **`messageType`** is set to ‘1’ and the message is protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
}
```

The field is defined as follows:

- **`Status`** - *Status* field in Table 35

6.5.1.3 Exception Handling

There may be unexpected exceptions during the Installation Setup Message processing as specified in section 5.5.1. The exception is classified into one of the following cases.

Case 1: The DRM Agent receives the InstallationSetupResponse with a *Status* other than *Success*. (i.e. the Handle was not stored by the SRM Agent)

Case 2: The Installation Setup Message processing is not completed for any reason other than Case 1.

When an exception occurs, the DRM Agent SHOULD immediately attempt to recover from it. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to the Operation Log when a new MAKE process is executed.

[Recovery Procedure – Cancellation of Move]

To cancel the Move transaction, the DRM Agent activates a recovery procedure for each type of exception as follows.

For Case 1, the Move is terminated without recovery. If the response contains *Duplicate Handle*, then the DRM Agent may start the Move transaction with a different Handle.

For Case 2, the DRM Agent sends the HandleRemovalRequest as specified in section 6.12.4 in order to remove the Handle. The Handle recorded in the Operation Log for this Move transaction MUST be used in this request. If the HandleRemovalResponse contains either *Success, Handle Not Found* or *Handle Not Removed* in the *Status* field, then the Move is terminated.

When the Move is terminated, the entry for the Move transaction is removed from the Operation Log.

If the Handle Removal Message processing is not completed for any reason other than those specified above, then the recovery procedure is aborted. The DRM Agent MAY resume the aborted recovery by sending the HandleRemovalRequest when a new MAKE process is executed. To resume the aborted recovery, the DRM Agent refers to the Operation Log as specified in section 5.5.2.

6.5.2 Rights Disablement in Device

6.5.2.1 Action Description

The DRM Agent disables the Rights. The disabled Rights cannot be used for the other purposes except the current Move transaction. After disabling the Rights, the DRM Agent continues with section 6.5.3.

6.5.2.2 Exception Handling

There may be unexpected exceptions as specified in section 5.5.1 when disabling Rights. This exception causes the disablement processing to not complete.

When the exception occurs, the DRM Agent SHOULD immediately attempt to recover from it. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to the Operation Log when a new MAKE process is executed. The recovery is same as the procedure for the exceptional Case 2 of the Installation Setup Message processing specified in section 6.5.1.3.

6.5.3 Rights Installation

6.5.3.1 Description of Messages

The DRM Agent sends the RightsInstallationRequest to install the Rights in the SRM. The fields of the request are defined in Table 37.

Table 37: Fields of RightsInstallationRequest

Fields	Protection Requirement	Description
Handle	Integrity & Confidentiality	Same as the Handle transmitted by the InstallationSetupRequest in Table 34. Refer to section 5.1.3.
REK	Integrity & Confidentiality	Refer to section 5.1.1.4
LAID	Integrity	Refer to 5.1.7. This contains the hash value of AssetIDs that are associated with the Rights.
Rights Information	Integrity	Refer to section 5.1.6

Upon receiving the RightsInstallationRequest, the SRM Agent installs the Rights in the SRM. For the installation, the SRM Agent MUST perform the following procedure:

1. Verify the integrity of the request fields
2. Decrypt the *Handle* and *REK* with the Session Key
3. Compare the *Handle* with the *Handle* in the InstallationSetupRequest
4. Install the *Rights Information* and *REK* at a space associated with the *Handle*.

The SRM Agent sends the RightsInstallationResponse to carry the result of the procedure. The fields of the response are defined in Table 38.

Table 38: Fields of RightsInstallationResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the RightsInstallationRequest message. The *Status* values are specified in Table 39.

Table 39: Status of Rights Installation Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Handle Not Found	The Handle in the request does not exist in the SRM.
Handles In-consistent	The Handle in this request is different from the *Handle* in the InstallationSetupRequest.
Not Enough Space	The size of *Rights Information* exceeds *Size of Rights* in Table 34.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response.

If no errors or exceptions (*Status* = *Success*), the DRM Agent continues with section 6.5.4.

6.5.3.2 Format of Messages

The message format (**`MessageBody`**) of the RightsInstallationRequest is specified as follows. The **`messageType`** is set to '0' and the message is protected by an HMAC.

```
HandleRek() {
    Handle()                        // Defined in E.2.4
    Rek()                           // Defined in E.2.5.5
}

EncryptedHandleRek() {
    // Contains the encrypted Handle and REK
    EncryptedData()                 // Defined in E.1
}

MessageBody() {
    EncryptedHandleRek()
    Laid()                          // Defined in Appendix E.3
    RightsInformation()             // Defined in Appendix E.2.5.4
}
```

The fields are defined as follows:

- **`HandleRek`** – *Handle* and *REK* fields in Table 37
- **`EncryptedHandleRek`** – Encrypted **`HandleRek`** with the current Session Key (SK)
- **`Laid`** – *LAID* field in Table 37
- **`RightsInformation`** – *Rights Meta Data*, *Rights Object Container*, *State Information* fields in Table 37 (Refer to Appendix E.2.5.4)

The message format (**MessageBody**) of the RightsInstallationResponse is specified as follows. The **messageType** is set to '1' and the message is protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
}
```

The field is defined as follows:

- **Status** - *Status* field in Table 38

6.5.3.3 Exception Handling

There may be unexpected exceptions during the Rights Installation Message processing as specified in section 5.5.1. The exception is classified into one of the following cases.

Case 1: The DRM Agent receives the RightsInstallationResponse containing a *Status* of *Handle Not Found* (This case will not happen if the Move transaction is properly executed as illustrated in Figure 12)

Case 2: The Rights Installation Message processing in this section is not completed for any reason other than Case 1.

When the exception occurs, the DRM Agent SHOULD immediately attempt to recover from it. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to the Operation Log when a new MAKE process is executed.

[Recovery Procedure – Cancellation of Move]

To cancel the Move transaction, the DRM Agent activates a default recovery procedure for each type of exception as follows.

For Case 1, the Move is terminated without recovery.

For Case 2, the DRM Agent sends the HandleRemovalRequest as specified in section 6.12.4 in order to remove the Handle. The Handle recorded in the Operation Log for this Move transaction MUST be used in this request. If the HandleRemovalResponse contains *Success* in the *Status* field, then the Move is terminated.

When the Move is terminated, the Rights in the source Device MUST be enabled (i.e. the Rights can be used for any purpose) and the entry for the Move transaction is removed from the Operation Log. In addition, if the Rights contain a <move> permission with a <count> constraint, then the current count value MUST be incremented.

In Case 2, if the HandleRemovalResponse contains a *Status* of either *Handle Not Removed* or *Handle Not Found*, then the DRM Agent continues the Move using the Rights Removal in Device processing defined in section 6.5.4. (**Note**: This result implies that the Rights were installed successfully in the SRM by the incomplete Rights Installation Message processing. In the case of *Handle Not Found*, after the installation, it implies the Rights were removed from the SRM or the corresponding Handle was updated to use the Rights.)

If the Handle Removal Message processing is not completed for any reason other than those specified above, then the recovery procedure is aborted. The DRM Agent MAY resume the aborted recovery by sending the HandleRemovalRequest when a new MAKE process is executed. To resume the recovery, the DRM Agent refers to the Operation Log as specified in section 5.5.2.

If the DRM Agent fails more than once to receive a proper *Status* from the Handle Removal Message processing during the recovery of the Rights Installation Message processing (i.e. fails to receive the response or fails to verify the integrity of the response) and then finally receives *Handle Not Found* in the *Status* field of the HandleRemovalResponse, it is possible that the Handle was successfully removed from the SRM by a previous incomplete Handle Removal Message processing. In this case, if the DRM Agent continues the Move with the Rights Removal in Device processing as specified in this section, then the User will lose the Rights.

The default behaviour is that the Move is terminated without further recovery procedures and the entry for the Move transaction is removed from the Operation Log (i.e. the Rights in the source Device stay in a disabled state). A trust model may define other procedures to handle the disabled Rights.

6.5.4 Rights Removal in Device

6.5.4.1 Action Description

The DRM Agent removes the Rights from the source Device permanently. When the Rights Removal in Device processing is completed, the Move is terminated and the entry for the Move transaction is removed from the Operation Log.

6.5.4.2 Exception Handling

There may be unexpected exceptions as specified in section 5.5.1 when removing Rights. The exception causes the removal processing to not complete.

When an exception occurs, the DRM Agent SHOULD immediately attempt to recover from it. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to the Operation Log when a new MAKE process is executed. The DRM Agent recovers from the exception by executing the Rights Removal in Device processing.

If the recovery fails, the DRM Agent MAY resume the recovery by removing the Rights from the Device when a new MAKE process is executed. To resume the recovery, the DRM Agent refers to the Operation Log as specified in section 5.5.2.

6.6 Movement of Rights from SRM to Device

Rights are Moved from an SRM to a Device as illustrated in Figure 13. As shown in Figure 13, this transaction is comprised of two request/response message pairs: Rights Retrieval message pair and Rights Removal message pair. The Rights Retrieval message pair SHOULD be followed by the Rights Removal message pair. If an SRM Agent receives any request message other than the RightsRemovalRequest following receipt of the RightsRetrievalRequest, the SRM Agent SHOULD return *Unexpected Request* in the *Status* field of the response message.

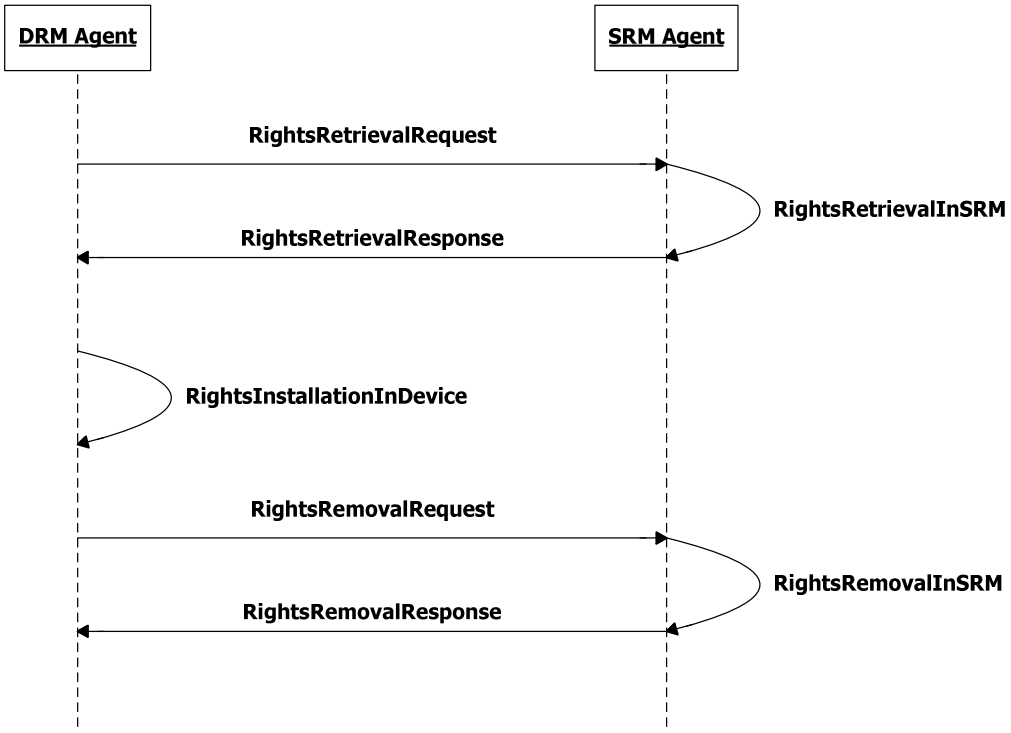

Figure 13: Sequence Diagram – Movement of Rights from SRM to Device

6.6.1 Rights Retrieval

6.6.1.1 Description of Messages

The DRM Agent sends the RightsRetrievalRequest to initiate the Move of the Rights from the SRM. The fields of the request are defined in Table 40.

Table 40: Fields of RightsRetrievalRequest

Fields	Protection Requirement	Description
Handle	Integrity	This identifies Rights that will be Moved from the SRM to the Device. Refer to section 5.1.3.
New Handle	Integrity & Confidentiality	*New Handle* is a 10 byte random value generated by the DRM Agent for this Move transaction.

Upon receiving the RightsRetrievalRequest, the SRM Agent MUST perform the following procedure:

1. Verify the integrity of the request fields
2. Find Rights corresponding to the *Handle*

3. If found, then decrypt the *New Handle* with the Session Key
4. Check if the SRM already has the same Handle with *New Handle*. If yes, the SRM Agent sets *Status* to *Duplicate Handle*. If no, overwrite the *Handle* in the SRM with the *New Handle*, and disable the Rights.

The SRM Agent sends the RightsRetrievalResponse to carry the result of the procedure. The fields of the response are defined in Table 41.

Table 41: Fields of RightsRetrievalResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the RightsRetrievalRequest message. The *Status* values are specified in Table 42. If *Status* contains any error, only this field is present in the RightsRetrievalResponse.
Rights Information	Integrity	Refer to section 5.1.6
REK	Integrity & Confidentiality	Refer to section 5.1.1.4

Table 42: Status of Rights Retrieval Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Handle Not Found	The Handle in the request does not exist in the SRM.
Duplicate Handle	The SRM already has the *New Handle* and its corresponding Rights.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent MUST perform the following procedure:

1. Verify the integrity of fields in the response
2. Decrypt *REK* with the Session Key

If no errors or exceptions (*Status* = *Success*), the DRM Agent continues with section 6.6.2.

6.6.1.2 Format of Messages

The message format (**`MessageBody`**) of the RightsRetrievalRequest is specified as follows. The **`messageType`** is set to ’0’ and the message is protected by an HMAC.

```
MessageBody() {
    Handle()                        // Defined in Appendix E.2.4
    EncryptedNewHandle()            // Defined in Appendix E.2.9
}
```

The fields are defined as follows:

- **`Handle`** –*Handle* field in Table 40
- **`EncryptedNewHandle`** – *New Handle* field in Table 40 encrypted with the current Session Key (SK)

The message format (**`MessageBody`**) of the RightsRetrievalResponse is specified as follows. The **`messageType`** is set to ‘1’ and the message is protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        RightsInformation()         // Defined in Appendix E.2.5.4
        EncryptedRek()              // Defined in Appendix E.2.7
    }
}
```

The fields are defined as follows:

- **`RightsInformation`** – *Rights Meta Data*, *Rights Object Container*, *State Information* fields in Table 41 (Refer to Appendix E.2.5.4)
- **`EncryptedRek`** – *REK* field in Table 41 (**`Rek`** in Appendix E.2.5.5) encrypted with the current Session Key (SK)
- **`Status`** - *Status* field in Table 41

6.6.1.3 Exception Handling

There may be unexpected exceptions during the Rights Retrieval Message processing as specified in section 5.5.1. The exception is classified into one of the following cases.

Case 1: The DRM Agent receives the RightsRetrievalResponse with a *Status* other than *Success*.

Case 2: The Rights Retrieval Message processing in this section is not completed for any reason other than Case 1.

When the exception occurs, the DRM Agent SHOULD immediately attempt to recover from it. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to the Operation Log when a new MAKE process is executed.

[Recovery Procedure – Cancellation of Move]

To cancel the Move transaction, the DRM Agent activates a recovery procedure for each type of exception as follows.

For Case 1, the Move is terminated without recovery. If the response contains *Duplicate Handle*, then the DRM Agent may start the Move transaction with a different Handle.

For Case 2, the DRM Agent sends the RightsEnablementRequest as specified in section 6.12.5 in order to enable the Rights. The *New Handle* recorded in the Operation Log for this Move transaction MUST be used in this request. If the RightsEnablementResponse contains a *Status* of either *Success* or *Handle Not Found*, then the Move is terminated.

When the Move is terminated, the entry for the Move transaction is removed from the Operation Log.

If the Rights Enablement Message processing is not completed for any reason other than those specified above, then the recovery procedure is aborted. The DRM Agent MAY resume the aborted recovery by sending the RightsEnablementRequest when a new MAKE process is executed. To resume the recovery, the DRM Agent refers to the Operation Log as specified in section 5.5.2.

6.6.2 Rights Installation in Device

6.6.2.1 Action Description

The DRM Agent performs the following procedure:

1. The RI-signature of the Rights SHOULD be verified (note that this requirement MAY not be required under certain trust models as described in section 5.1.2). If the verification fails, the DRM Agent MUST re-enable the Rights on the SRM as described in section 6.12.5 and terminate the Move transaction.

2. In case of stateful Rights, the State Information SHOULD be checked that it is consistent with the **<rights>** element[1]. If the State Information is inconsistent, the DRM Agent MUST re-enable the Rights on the SRM as described in section 6.12.5 and terminate the Move transaction.

3. The DRM Agent MUST check that the Rights have a **<move>** permission. If there is no **<move>** permission, the DRM Agent MUST re-enable the Rights on the SRM as described in section 6.12.5 and terminate the Move transaction.

4. The DRM Agent MUST check if the **<move>** permission has a **<count>** constraint. If there is no **<count>** constraint, continue with step 5. Otherwise, the DRM Agent MUST check if the remaining count is not zero. If not zero, the DRM Agent MUST decrement the remaining count and continue with step 5. Otherwise (the remaining count is zero), the DRM Agent MUST re-enable the Rights on the SRM as described in section 6.12.5 and terminate the Move transaction.

5. The DRM Agent MUST check if the **<move>** permission has a **<system>** constraint. If there is no **<system>** constraint, continue with step 6. Otherwise, the DRM Agent MUST check if the **<context>** child element of the **<system>** constraint identifies the SRM protocol. If it does, continue with step 6. Otherwise, the DRM Agent MUST re-enable the Rights on the SRM as described in section 6.12.5 and terminate the Move transaction.

6. Install the Rights with the following conditions:

   A. Rights received via the Move protocol SHALL NOT be rejected based on the content of any DRM V2.0 replay cache. The replay cache is specified in section 9.4 of [OMADRMv2.0].

   B. The Rights SHALL NOT be installed if a Rights Object with the same ROID is already installed. If so, then the DRM Agent MUST re-enable the Rights on the SRM as described in section 6.12.5 and terminate the Move transaction.

   C. If the <GUID, RITS> pair for the Rights is already in the Move Cache, then this <GUID, RITS> pair MUST be removed from the Move Cache after installation of the Rights. (See section 9).

After the Rights installation, the DRM Agent continues with section 6.6.3.

6.6.2.2 Exception Handling

There may be unexpected exceptions as specified in section 5.5.1 when installing Rights. The exception causes the installation to not complete.

When an exception occurs, the DRM Agent SHOULD immediately attempt to recover from it. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to the Operation Log when a new MAKE process is executed. The recovery is same as the procedure for the exception in Case 2 of the Rights Retrieval Message processing specified in section 6.6.1.3.

6.6.3 Rights Removal

6.6.3.1 Description of Messages

The DRM Agent executes the Rights Removal Message processing as specified in section 6.12.6 in order to remove the original Rights from the SRM.

The *Handle* in the RightsRemovalRequest MUST be identical to the *New Handle* in the previous RightsRetrievalRequest specified in section 6.6.1.1.

[1] For example, if the Rights have a <count> constraint of 5 but the State Information indicates that the remaining count is 10, the State Information is inconsistent.

When the Rights Removal Message processing is completed (*Status* = *Success*), the Move is terminated and the entry for the Move transaction is removed from the Operation Log.

6.6.3.2 Format of Messages

Refer to section 6.12.6.2.

6.6.3.3 Exception Handling

There may be unexpected exceptions during the Rights Removal Message processing as specified in section 5.5.1. The exception is classified into one of the following cases.

Case 1: The DRM Agent receives the RightsRemovalResponse containing a *Status* of *Handle Not Found.* (This case will not happen if the Move transaction is properly executed as illustrated in Figure 13)

Case 2: The Rights Removal Message processing is not completed for any reason other than Case 1.

When an exception occurs, the DRM Agent SHOULD immediately attempt to recover from it. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to the Operation Log when a new MAKE process is executed.

[Recovery Procedure – Completion of Move]

To complete the Move transaction, the DRM Agent activates a recovery procedure for each exception type as follows.

For Case 1, the Move is terminated without recovery.

For Case 2, the DRM Agent sends the RightsRemovalRequest as specified in section 6.12.6 in order to remove the Rights from the SRM. The *New Handle* recorded in the Operation Log for this Move transaction MUST be used in this request. If the RightsRemovalResponse contains a *Status* of either *Success* or *Handle Not Found*, then the Move is terminated.

When the Move is terminated, the entry for the Move transaction is removed from the Operation Log.

If the Rights Removal Message processing for the exception recovery is not completed for any reason other than those specified above, then the recovery procedure is aborted. The DRM Agent MAY resume the aborted recovery by sending the RightsRemovalRequest when a new MAKE process is executed. To resume the recovery, the DRM Agent refers to the Operation Log as specified in section 5.5.2.

6.7 Local Rights Consumption

[Initiation of Local Rights Consumption]

To use a DRM Content by consuming its associated Rights, the DRM Agent may collect Rights Information associated with the DRM Content from the SRM (Refer to section 6.12.2 and section 6.12.3). If there are more than one associated Rights in the SRM, the DRM Agent may perform it multiple times.

The DRM Agent selects one of the Rights for consumption by referring to permissions and constraints in the Rights Information (refer to section 6.7.1). After the Rights selection, the DRM Agent reads the REK of the selected Rights and disables the Rights (refer to section 6.7.2).

[Local Rights Consumption]

A DRM Content is used by consuming Rights from the SRM as specified in section 6.7.3. Local Rights Consumption is illustrated in Figure 14.

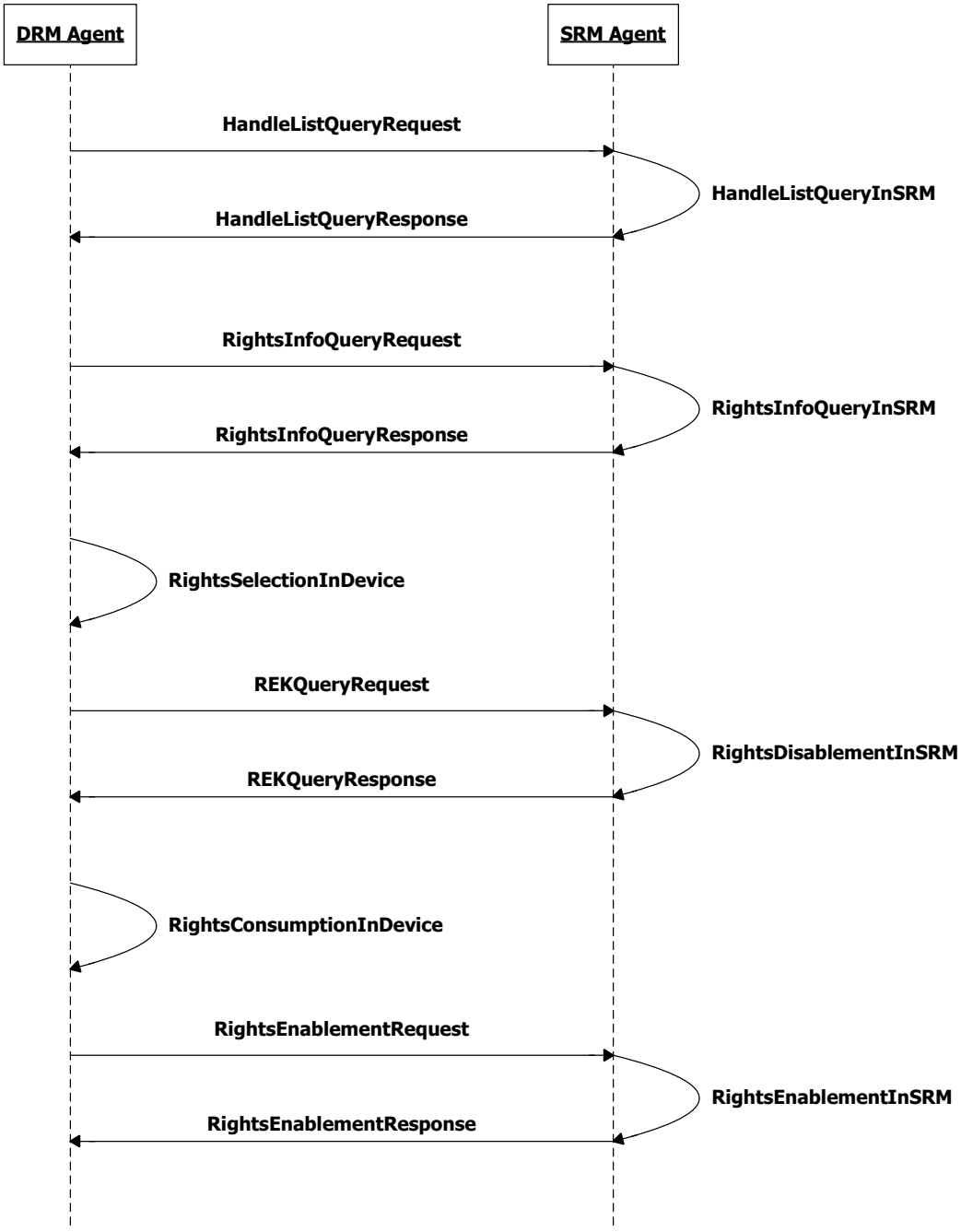

Figure 14: Sequence Diagram – Local Rights Consumption

6.7.1 Rights Selection in Device

For a particular DRM Content, both the SRM and the Device may have Rights associated with it. Then the DRM Agent selects one of the Rights. It is assumed that the DRM Agent may read associated Rights Information from the SRM as specified in section 6.12.2 and section 6.12.3. The selection may be achieved by the DRM Agent itself according to the rights evaluation order as specified in [OMADRMv2.0] or may need User interaction.

If the DRM Agent selects Rights from the Device, the consumption of the Rights is performed as specified in [OMADRMv2].

If the DRM Agent selects Rights from the SRM, then the DRM Agent continues with the REK Query Message processing as specified in section 6.7.2.

6.7.2 REK Query

The DRM Agent receives the REK of Rights from the SRM Agent as illustrated in Figure 15.

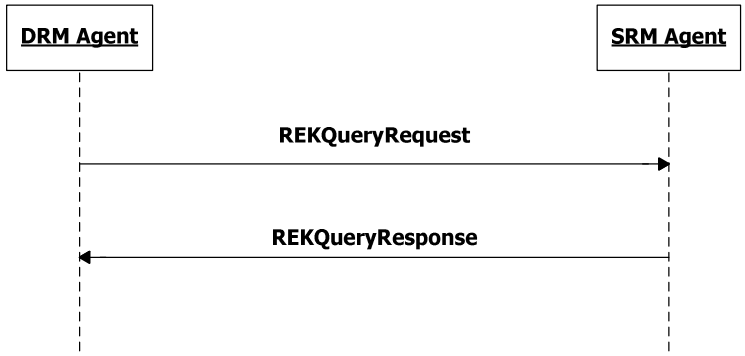

Figure 15: Sequence Diagram – REK Query

6.7.2.1 Description of Messages

The DRM Agent sends the REKQueryRequest for the SRM Agent to read an REK and disable its corresponding Rights in the SRM. The fields of the request are defined in Table 43.

Table 43: Fields of REKQueryRequest

Fields	Protection Requirement	Description
Handle	Integrity	This identifies Rights whose REKs will be transferred from the SRM to the Device. Refer to section 5.1.3.
New Handle	Integrity & Confidentiality	*New Handle* is a 10 byte random value generated by the DRM Agent for this Local Rights Consumption transaction.

Upon receiving the REKQueryRequest, the SRM Agent MUST performs the following procedure:

1. Verify the integrity of the request fields
2. Find Rights corresponding to the *Handle*
3. If found, then decrypt the *New Handle* with the Session Key, read REK of the Rights, overwrite the *Handle* in the SRM with the *New Handle*, and disable the Rights
4. Check if the SRM already has the same Handle with *New Handle*. If yes, the SRM Agent sets *Status* to *Duplicate Handle*. If no, read REK of the Rights, overwrite the *Handle* in the SRM with the *New Handle*, and disable the Rights

A trust model may decide that the disabled Rights are enabled automatically when a new Device – SRM Hello processing (specified in section 6.1) is executed. Default behaviour is that the disabled Rights SHALL NOT be enabled without a request from the DRM Agent that disabled the Rights.

The SRM Agent sends the REKQueryResponse to carry the result of the procedure. The fields of the response are defined in Table 44.

Table 44: Fields of REKQueryResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the REKQueryRequest message. The *Status* values are specified in Table 45. If *Status* contains any error, only this field is present in the REKQueryResponse.
REK	Integrity & Confidentiality	Refer to section 5.1.1.4

Table 45: Status of REK Query Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Duplicate Handle	The SRM already has the New Handle and its corresponding Rights.
Handle Not Found	The Handle in the request does not exist in the SRM.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent MUST perform the following procedure:

1. Verify the integrity of fields in the response
2. Decrypt *REK* with the Session Key

If no errors or exceptions (*Status* = *Success*), the DRM Agent completes the REK Query Message processing.

6.7.2.2 Format of Messages

The message format (**`MessageBody`**) of the REKQueryRequest is specified as follows. The **`messageType`** is set to ’0’ and the message is protected by an HMAC.

```
MessageBody() {
    Handle()                        // Defined in Appendix E.2.4
    EncryptedNewHandle()            // Defined in Appendix E.2.9
}
```

The fields are defined as follows:

- **`Handle`** –*Handle* field in Table 43
- **`EncryptedNewHandle`** – *New Handle* field from Table 43 encrypted with the current Session Key (SK)

The message format (**`MessageBody`**) of the REKQueryResponse is specified as follows. The **`messageType`** is set to ‘1’ and the message is protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        EncryptedRek()              // Defined in Appendix E.2.7
    }
}
```

The fields are defined as follows:

- **`EncryptedRek`** – *REK* field in Table 44 (**`Rek`** in Appendix E.2.5.5) encrypted with the current Session Key (SK)
- **`Status`** - *Status* field in Table 44

6.7.2.3 Exception Handling

There may be unexpected exceptions during the REK Query Message processing as specified in section 5.5.1. The exception is classified into one of the following cases.

Case 1: The DRM Agent receives the REKQueryResponse with a *Status* other than *Success*.

Case 2: The REK Query Message processing is not completed for any reason other than Case 1.

When an exception occurs, the DRM Agent SHOULD immediately attempt to recover from it. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to the Operation Log when a new MAKE process is executed.

[Recovery Procedure]

The DRM Agent activates a recovery procedure for each exception type as follows.

For Case 1, the Local Rights Consumption is terminated without recovery. If the REKQueryResponse contains a *Status* of *Handle Not Found*, the DRM Agent may restart the Local Rights Consumption with the Rights Selection in Device processing in section 6.7.1.

For Case 2, the DRM Agent sends the RightsEnablementRequest as specified in section 6.12.5 in order to enable the Rights and to update the State Information, if necessary. The *New Handle* recorded in the Operation Log for this Local Rights Consumption transaction MUST be used in this request. If the RightsEnablementResponse contains a *Status* of either *Success* or *Handle Not Found*, then the Local Rights Consumption is terminated.

When the Local Rights Consumption transaction is terminated, the entry for the transaction is removed from the Operation Log.

If the Rights Enablement Message processing is not completed for any reason other than those specified above, then the recovery procedure is aborted. The DRM Agent MAY resume the aborted recovery by sending the RightsEnablementRequest when a new MAKE process is executed. To resume the recovery, the DRM Agent refers to the Operation Log as specified in section 5.5.2.

6.7.3 Rights Consumption and Release

The DRM Agent performs the following steps before locally consuming Rights in the SRM in order to use an associated DRM Content:

In case of stateful Rights, the DRM Agent SHOULD verify that the State Information is consistent with the **<rights>** element[2]. If the State Information is inconsistent, the DRM Agent MUST re-enable the Rights on the SRM as described in section 6.12.5 and terminate local consumption.

The DRM Agent SHOULD verify the RI signature over the **<rights>** element (note that this requirement MAY not be required under certain trust models as described in section 5.1.2). If the signature verification fails, the DRM Agent MUST re-enable the Rights on the SRM as described in section 6.12.5 and terminate local consumption.

After the DRM Agent retrieves the REK (as specified in section 6.7.2), the DRM Agent SHALL locally consume the Rights as if the Rights are locally installed in the Device, updating the state as specified in [OMADRMv2.0]. After local consumption, the DRM Agent SHALL update the State Information (for stateful Rights) in the SRM when it releases the Rights as specified in this section.

Note that a trust model may define different timing of the Rights Enablement Message activation for each constraint. Default behaviour is that the Rights Enablement Message processing is executed after consumption.

6.7.3.1 Description of Messages

The DRM Agent releases the Rights using Rights Enablement Message as specified in section 6.12.5 and the REK MUST be removed after Rights Enablement Message processing is successfully executed.

The State Information field MUST be present when releasing Stateful Rights.

6.7.3.2 Format of Messages

Refer to section 6.12.5.2.

6.7.3.3 Exception Handling

There may be unexpected exceptions during the Rights Enablement Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response, receives an error in the *Status* field, or fails to verify the message integrity then the DRM Agent regards it as an exception.

When an exception occurs, the DRM Agent SHOULD immediately attempt to recover from it. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to the Operation Log when a new MAKE process is executed.

[Recovery Procedure]

The DRM Agent sends the RightsEnablementRequest. The *New Handle* recorded in the Operation Log for this Local Rights Consumption transaction MUST be used in this request. If the RightsEnablementResponse contains a *Status* of either *Success* or *Handle Not Found*, then the Local Rights Consumption is terminated.

When the Local Rights Consumption transaction is terminated, the entry for the transaction is removed from the Operation Log.

If the Rights Enablement Message processing for the exception recovery is not completed for any reason other than those specified above, then the recovery procedure is aborted. The DRM Agent MAY resume the aborted recovery by sending the RightsEnablementRequest when a new MAKE process is executed. To resume the recovery, the DRM Agent refers to the Operation Log as specified in section 5.5.2.

[2] For example, if the Rights have a <count> constraint of 5 but the State Information indicates that the remaining count is 10, the State Information is inconsistent.

6.8 Direct Provisioning of Rights to the SRM

The protocols specified in this section provide necessary functions that are used for Direct Provisioning of Rights to the SRM transaction.

The Figure 16 describes overall flow for how Rights are downloaded and installed to the SRM. As shown in Figure 16, this transaction is comprised of the 2-pass ROAP with trigger between RI and DRM Agent and three request/response message pairs between DRM Agent and SRM Agent: Signature Query message pair, Provisioning Setup and Rights Provisioning message pair.

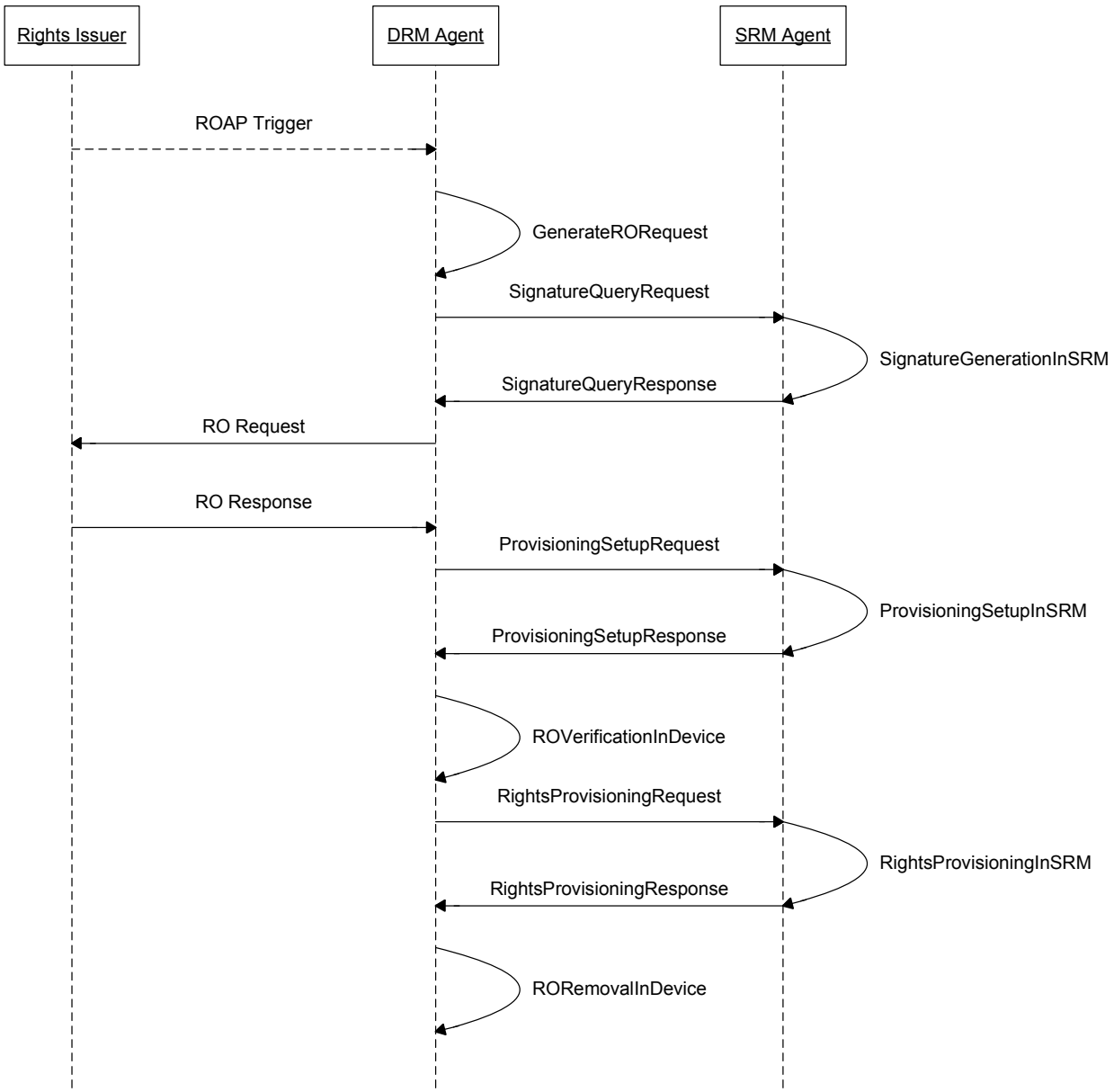

Figure 16: Sequence Diagram – Direct Provisioning of Rights to the SRM

The description for overall flow is as follows:

1. The RI MAY send an ROAP RO Acquisition Trigger to a DRM Agent. If the DRM Agent receives the RO Acquisition Trigger, it processes the trigger message, as specified in the section 6.8.1. The DRM Agent MAY also initiate the Direct Provisioning transaction on its own accord.
2. The DRM Agent SHALL generate the RO Request message, as specified in the section 6.8.2.
3. The DRM Agent SHALL execute the SignatureQuery protocol with the SRM Agent, as specified in the section 6.8.3.
4. The DRM Agent SHALL execute ROAP RO Acquisition protocol with the RI, as specified in the section 6.8.5
5. The DRM Agent SHALL execute ProvisioningSetup protocol with the SRM Agent, as specified in the section 6.8.6
6. The DRM Agent SHALL verify the RO for the SRM Agent, as specified in the section 6.8.6.
7. The DRM Agent SHALL execute RightsProvisioning protocol with the SRM Agent, as specified in the section 6.8.7.
8. The DRM Agent SHALL remove corresponding RO, as specified in the section 6.8.8.

6.8.1 RO Acquisition Trigger

6.8.1.1 Action Description

When the DRM Agent receives an RO Acquisition Trigger, it processes the trigger message as follows:

1. It checks if the trigger is for Direct Provisioning of Rights to the SRM. The DRM Agent SHALL check the presence of the <trustAnchorAndsrmIDPair> element in the trigger, and if the element is present the DRM Agent determines that the trigger is for Direct Provisioning of Rights to the SRM.
2. If the trigger has the <trustAnchorAndsrmIDPair> element, the DRM Agent SHALL compare the <trustAnchor> element and <srmID> element in the <trustAnchorAndsrmIDPair> element with the Trust Anchor and SRMID pair of the SAC Context in an SRM attached Device. There are two possible outcomes of this comparison:

   A. If the <trustAnchor> element and the <srmID> element pair in the <trustAnchorAndsrmIDPair> element matches with any Trust Anchor and SRMID pair of the SAC Context in the DRM Agent, continues with section 6.8.2.

   B. If none of the <trustAnchor> and <srmID> element pairs match a Trust Anchor and SRMID pair of the SAC Context in the DRM Agent, the DRM Agent SHALL perform the following procedure:

      i. If any <trustAnchorAndsrmIDPair> element in the trigger matches one of the Trust Anchor and SRM ID pairs in the *Trust Anchor And SRM ID Pair List* returned in the SRMHelloResponse message, the DRM Agent SHALL initiate the MAKE procedure. The Authentication Request SHALL include the Trust Anchor that was supported by DRM Agent and included in the trigger. After receiving the Authentication Request, the SRM Agent SHALL perform the Authentication procedure starting from checking the Trust Anchor in the Authentication Request as follows:

         - If the SRM Agent supports the Trust Anchor received in Authentication Request, the SRM Agent SHALL continue the remaining Authentication procedure (see the section 6.2.1.1). If the Authentication procedure is completed successfully, the SRM Agent SHALL send the Authentication Response to the DRM Agent with the 'Success' status and proceed to the Key Exchange procedure (see the section 6.2.2). If the Key Exchange procedure is completed successfully, the DRM Agent continues with section 6.8.2**,** otherwise it determines this transaction as failure and terminates the Direct Provisioning transaction.
         - If the SRM Agent does not support the Trust Anchor received in Authentication Request, the SRM Agent SHALL send the Authentication Response to the DRM Agent with the error ('Trust Anchor Not Supported') status. Upon receiving an Authentication Response with error status, the DRM Agent considers this transaction as failed and terminates the Direct Provisioning transaction.

ii. If no <trustAnchorAndsrmIDPair> element in the trigger matches any of the Trust Anchor and SRM ID pairs in the *Trust Anchor And SRM ID Pair List* returned in the SRMHelloResponse message, the DRM Agent considers this transaction as failed and terminates the Direct Provisioning transaction.

3. If the trigger does not have the <trustAnchorAndsrmIDPair> element, the DRM Agent determines this trigger is not related to the Direct Provisioning transaction.

6.8.2 Generate RO Request

6.8.2.1 Action Description

The DRM agent SHALL generate the RO Request message as follows:

1. The request message SHALL include the SRM ID instead of a Device ID. The SRM ID SHALL correspond with the selected Trust Anchor in the section 6.8.1.
2. The request message SHALL NOT include the signature, so that the SRM will generate the signature over the request message without DRM Agent's signature.
3. The request message SHALL include a Trust Anchor in the extension element, where the Trust Anchor is selected between the DRM Agent and the SRM Agent according to the section 6.8.1.
4. The request message MUST include the SRM's Certificate Chain instead of a Device's Certificate Chain.
5. The Generated RO Request message SHALL be canonicalised by the DRM Agent.
6. The DRM Agent SHALL store the generated canonicalised RO Request message until the DRM Agent sends the RO Request message to the RI as specified in the section 6.8.4.

6.8.3 Signature Query

To confirm an RO acquisition request from an SRM, the DRM Agent requests a signature of the SRM Agent. The Signature Query Message processing is used to get a digital signature of SRM Agent on a RO Request message which was generated by DRM Agent.

6.8.3.1 Description of Messages

The DRM Agent sends the SignatureQueryRequest to get the Signature of the SRM on the RO Request message. The field of the request is defined in Table 46.

Table 46: Fields of SignatureQueryRequest

Fields	Protection Requirement	Description
RO Request	Integrity	RO Request message that was generated by the DRM Agent. The RO Request message needs a signature of an SRM for confirmation of the RO acquisition request. The RO Request is already in canonicalised form
Signature Scheme	Integrity	This field contains a negotiated signature scheme between the RI and the Device.

Upon receiving the SignatureQueryRequest, the SRM Agent SHALL proceeds as follows:

- The SRM Agent verifies that the RO Request starts with "<roap:roRequest>".
- If the verification fails, the SRM Agent SHALL return a SignatureQueryResponse with the Status set to "Not an RO Request".
- Otherwise, the signature is calculated in accordance with the rules of the received signature scheme.

- The signature is calculated in accordance with the rules of the received signature scheme.

The SRM Agent sends the SignatureQueryResponse to carry the result of the action. The fields of the response are defined in Table 47.

Table 47: Fields of SignatureQueryResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the SignatureQueryRequest message. The *Status* values are specified in Table 48. If *Status* contains any error, only this field is present in the SignatureQueryResponse.
Signature of RO Request	Integrity	This field contains a signature of the SRM.

Table 48: Status of Signature Query Message

Status Value	Description
Success	The request was successfully processed.
Signature Scheme Not Supported	This signature scheme is not supported by the SRM Agent.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Not an RORequest	The RO Request to be signed does not start with "<roap:roRequest>".
Unknown Error	Other errors

If the value of the *Status* field within a valid SignatureQueryResponse message is not *Success*, then the DRM Agent determines this transaction as failure and terminates the Direct Provisioning transaction.

If no errors or exceptions (*Status* = *Success*), the DRM Agent continues with section 6.8.4.

6.8.3.2 Format of Messages

The message format (**`MessageBody`**) of the SignatureQueryRequest is specified as follows. The **`messageType`** is set to '0' and the message is not protected by an HMAC.

```
RORequest(){
    OctetString16()                    // Defined in Appendix E.1
}


SignatureScheme(){
    Algorithm()                        //  Defined in Appendix E.1
}


MessageBody() {
    RORequest()
    SignatureScheme()
}
```

The fields are defined as follows:

- **`RORequest`** – RO Request field in Table 46.

- **SignatureScheme** – negotiated Signature Scheme field in Table 46 between the RI and the Device.

The message format (**MessageBody**) of the SignatureResponse is specified as follows. The **messageType** is set to '1' and the message is not protected by an HMAC.

```
SignatureOfRORequest(){
    OctetString16()                // Defined in Appendix E.1
}

MessageBody() {
    Status()                       // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        SignatureOfRORequest()
    }
}
```

The fields are defined as follows:

- **status** - *Status* field in Table 47.
- **SignatureOfRORequest** – *Signature of RO Request* field in Table 47.

6.8.3.3 Exception Handling

There may be an unexpected exception during the Signature Query Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response or finds an error by referring to the *Status*, then the DRM Agent regards it as an exception and terminates the Signature Query Message processing. The User may be informed of the exception.

6.8.4 RO Acquisition between RI and DRM Agent

6.8.4.1 Action Description

Upon receiving the SignatureQueryResponse, the DRM Agent SHALL insert the received signature on RO Request message into the RO Request message that was stored in the Device and send the resulting RO Request message to the RI.

Upon receiving the RO Request message, the RI SHALL generate the RO Response message and send it to the DRM Agent as follows:

1. The response message SHALL include the SRMID instead of a Device ID.
2. The Rights Objects (in the form of **<ProtectedRO>** elements) in response message SHALL be cryptographically bound to the SRM Agent, so that the concatenation of K_{MAC} and K_{REK} in the <ProtectedRO> element is encrypted with the SRM Agent's public key.

If the DRM Agent receives an RO Response message with "Success" as the status, the DRM Agent MUST perform the following procedure:

1. Verify the signature of the RO Response message
2. If the signature is valid, the DRM Agent SHALL check the device ID in the RO Response message. If the device ID matches an SRM ID inserted in the Device, then the DRM Agent SHALL extract the Rights from the protected RO in the RO Response message and the DRM Agent continues with section 6.8.5. If the Rights are stateful, the State Information SHALL be generated by the DRM Agent and it SHALL be included within the Rights

Information in RightsProvisioningRequest. (for information on the RightsProvisioningRequest, see the section 6.8.7)

The format of the RO Request message and the RO Response message is same as specified in DRM 2.0 [OMADRMv2.0].

6.8.5 Provisioning Setup

6.8.5.1 Description of Messages

The DRM Agent sends the ProvisioningSetupRequest to initiate a Direct Provisioning of Rights to the SRM. The fields of the request are defined in Table 49.

Table 49: Fields of ProvisioningSetupRequest

Fields	Protection Requirement	Description
Handle	Integrity & Confidentiality	The *Handle* identifies the Rights while stored in the SRM. It is a 10 byte random value generated by the DRM Agent for this Provisioning transaction. Refer to section 5.1.3.
Size of Rights	Integrity	Size of the Rights in bytes. This informs the SRM Agent about the size of the Rights that will be installed in the SRM as specified in section 6.5.3 *Size of Rights* = Length of **`RightsInformation`**. **`RightsInformation`** is specified in section 6.5.3.2.
Wrapped Key Material	Integrity	The *Wrappted Key Material* consists of a wrapped concatenation of a MAC Key, K_{MAC} and an RO Encryption Key, K_{REK}. The Wrapped Key Material is same as the *C* value as specified in DRM v2.1 [OMADRMv2.1] section 7.2.1

Upon receiving the ProvisioningSetupRequest, the SRM Agent MUST perform the following procedure:

1. Verify the integrity of the fields
2. Decrypt the *Handle* with the Session Key
3. The SRM Agent MUST check if the SRM already has the same Handle. If yes, the SRM Agent sets *Status* to *Duplicate Handle* and the SRM Agent sends the ProvisioningSetupResponse as described below.
4. The SRM Agent checks if the SRM has space for the new Rights. If not, the SRM Agent sets *Status* to *Not Enough Space*. Otherwise, the SRM Agent stores the *Handle* in the SRM securely. The *Handle* is not included in the Handle List until the Provisioning transaction is completed.
5. The SRM Agent MUST check Wrapped K_{MAC} and K_{REK}. The SRM 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*:

$$C_1 \mid C_2 = C$$

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

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

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

Using *Z*, the SRM Agent 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

6. The SRM Agent MUST install the *REK* (K_{REK}) at a space associated with the *Handle*.

The SRM Agent sends the ProvisioningSetupResponse to carry the result of the procedure. The fields of the response are defined in Table 50.

Table 50: Fields of ProvisioningSetupResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the ProvisioningSetupRequest message. The *Status* values are specified in Table 51.
MAC Key	Integrity & Confidentiality	The encrypted K_{MAC} from the ProvisioningSetupRequest. It is used for key confirmation of the RO Response message carrying *REK*.

Table 51: Status of Provisioning Setup Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Duplicate Handle	The SRM already has the Handle and its corresponding Rights.
Not Enough Space	The SRM does not have enough space to store Rights having the same size as the *Size of Rights*.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response.

If no errors or exceptions (*Status* = *Success*), the DRM Agent continues with section 6.8.6.

6.8.5.2 Format of Messages

The message format (**`MessageBody`**) of the ProvisioningSetupRequest is specified as follows. The **`messageType`** is set to ’0’ and the message is protected by an HMAC.

```
WrappedKeyMaterial() {
    //Contains the encrypted Wrapped Key Material
    OctetString16()
}


MessageBody() {
    EncryptedHandle()               // Defined in Appendix E.2.8
    sizeOfRights            16      uimsbf
    WrappedKeyMaterial()
}
```

The fields are defined as follows:

- **`sizeOfRights`** –*Size Of Rights* field in Table 49
- **`EncryptedHandle`** –**`Handle`** encrypted with the current Session Key (SK)

`WrappedKeyMaterial` – **`Wrapped Key Matrial`** field in Table 49

The message format (**`MessageBody`**) of the ProvisioningSetupResponse is specified as follows. The **`messageType`** is set to ‘1’ and the message is protected by an HMAC.

```
EncryptedMACKey() {
    // Contains the encrypted MAC Key
    EncryptedData()                 // Defined in E.1
}


MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if(Status == 0){
        EncryptedMACKey()
    }
}
```

The field is defined as follows:

- **`Status`** - *Status* field in Table 50
- **`EncryptedMACKey`** –$\mathbf{K_{MAC}}$ from the ProvisioningSetupRequest, encrypted with the current Session Key (SK)

6.8.5.3 Exception Handling

There may be unexpected exceptions during the Provisioning Setup Message processing as specified in section 5.5.1. The exception is classified into one of the following cases.

Case 1: The DRM Agent receives the ProvisioningSetupResponse with a *Status* other than *Success*. (i.e. the Handle was not stored by the SRM Agent)

Case 2: The Provisioning Setup Message processing is not completed for any reason other than Case 1.

When an exception occurs, the DRM Agent SHOULD immediately attempt to recover from it. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to the Operation Log when a new MAKE process is executed.

[Recovery Procedure – Cancellation of Provisioning]

To cancel the Provisioning transaction, the DRM Agent activates a recovery procedure for each type of exception as follows.

For Case 1, the Provisioning is terminated without recovery. If the response contains *Duplicate Handle*, then the DRM Agent may start the Provisioning transaction with a different Handle.

For Case 2, the DRM Agent sends the HandleRemovalRequest as specified in section 6.12.4 in order to remove the Handle. The Handle recorded in the Operation Log for this Provisioning transaction MUST be used in this request. If the HandleRemovalResponse contains either *Success*, *Handle Not Found* or *Handle Not Removed* in the *Status* field, then the Provisioning is terminated.

When the Provisioning is terminated, the entry for the Provisioning transaction is removed from the Operation Log.

If the Handle Removal Message processing is not completed for any reason other than those specified above, then the recovery procedure is aborted. The DRM Agent MAY resume the aborted recovery by sending the HandleRemovalRequest when a new MAKE process is executed. To resume the aborted recovery, the DRM Agent refers to the Operation Log as specified in section 5.5.2.

6.8.6 RO Verification in Device

6.8.6.1 Action Description

The DRM Agent MUST verify the MAC value on the RO for SRM using the MAC Key in Provisioning Setup Response message.

6.8.7 Rights Provisioning

6.8.7.1 Description of Messages

The DRM Agent sends the RightsProvisioningRequest to install the Rights in the SRM. The fields of the request are defined in Table 52.

Table 52: Fields of RightsProvosioningRequest

Fields	Protection Requirement	Description
Handle	Integrity & Confidentiality	Same as the Handle transmitted by the ProvisioningSetupRequest in Table 50. Refer to section 6.8.5.1.
LAID	Integrity	Refer to 5.1.7. This contains the hash value of AssetIDs that are associated with the Rights.
Rights Information	Integrity	Refer to section 5.1.6

Upon receiving the RightsProvisioningRequest, the SRM Agent installs the Rights in the SRM. For the installation, the SRM Agent MUST perform the following procedure:

1. Verify the integrity of the request fields
2. Decrypt the *Handle* with the Session Key
3. Compare the *Handle* with the *Handle* in the ProvisioningSetupRequest
4. Install the *Rights Information* at a space associated with the *Handle*.

The SRM Agent sends the RightsProvisionResponse to carry the result of the procedure. The fields of the response are defined in Table 53.

Table 53: Fields of RightsProvisioningResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the RightsProvisioningRequest message. The *Status* values are specified in Table 54.

Table 54: Status of Rights Provisioning Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Handle Not Found	The Handle in the request does not exist in the SRM.
Handles In-consistent	The Handle in this request is different from the *Handle* in the ProvisioningSetupRequest.
Not Enough Space	The size of *Rights Information* exceeds *Size of Rights* in Table 49.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.

Status Value	Description
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response.

If no errors or exceptions (*Status* = *Success*), the DRM Agent continues with section 6.8.8.

6.8.7.2 Format of Messages

The message format (**`MessageBody`**) of the RightsProvisioningRequest is specified as follows. The **`messageType`** is set to '0' and the message is protected by an HMAC.

```
EncryptedHandle() {
    // Contains the encrypted Handle
    EncryptedData()                 // Defined in E.1
}

MessageBody() {
    EncryptedHandle()
    Laid()                          // Defined in Appendix E.3
    RightsInformation()             // Defined in Appendix E.2.5.4
}
```

The fields are defined as follows:

- **`EncryptedHandle`** – Encrypted **`Handle`** with the current Session Key (SK)
- **`Laid`** – *LAID* field in Table 52
- **`RightsInformation`** – *Rights Meta Data*, *Rights Object Container*, *State Information* fields in Table 52 (Refer to Appendix E.2.5.4)

The message format (**`MessageBody`**) of the RightsProvisioningResponse is specified as follows. The **`messageType`** is set to '1' and the message is protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
}
```

The field is defined as follows:

- **`Status`** - *Status* field in Table 53

6.8.7.3 Exception Handling

There may be unexpected exceptions during the Rights Provisioning Message processing as specified in section 5.5.1. The exception is classified into one of the following cases.

Case 1: The DRM Agent receives the RightsProvisioningResponse containing a *Status* of *Handle Not Found* (This case will not happen if the Provisioning transaction is properly executed as illustrated in Figure 12)

Case 2: The Rights Provisioning Message processing in this section is not completed for any reason other than Case 1.

When the exception occurs, the DRM Agent SHOULD immediately attempt to recover from it. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to the Operation Log when a new MAKE process is executed.

[Recovery Procedure – Cancellation of the Provisioning transaction]

To cancel the Provisioning transaction, the DRM Agent activates a default recovery procedure for each type of exception as follows.

For Case 1, the Provisioning transaction is terminated without recovery.

For Case 2, the DRM Agent sends the HandleRemovalRequest as specified in section 6.12.4 in order to remove the Handle. The Handle recorded in the Operation Log for this Provisioning transaction MUST be used in this request. If the HandleRemovalResponse contains *Success* in the *Status* field, then the Provisioning transaction is terminated and the entry for the Provisioning transaction is removed from the Operation Log. The DRM Agent MAY retry to install the Rights on the SRM, by sending a new ProvisioningSetupRequest message.

If the HandleRemovalResponse contains a *Status* of either *Handle Not Removed* or *Handle Not Found*, then the DRM Agent continues the Provisioning transaction using the Rights Removal in Device processing defined in section 6.5.4. (**Note**: This result implies that the Rights were installed successfully in the SRM by the incomplete Rights Provisioning Message processing. In the case of *Handle Not Found*, after the installation, it implies the Rights were removed from the SRM or the corresponding Handle was updated to use the Rights.)

If the Handle Removal Message processing is not completed for any reason other than those specified above, then the recovery procedure is aborted. The DRM Agent MAY resume the aborted recovery by sending the HandleRemovalRequest when a new MAKE process is executed. To resume the recovery, the DRM Agent refers to the Operation Log as specified in section 5.5.2.

If the DRM Agent fails more than once to receive a proper *Status* from the Handle Removal Message processing during the recovery of the Rights Provisioning Message processing (i.e. fails to receive the response or fails to verify the integrity of the response) and then finally receives *Handle Not Found* in the *Status* field of the HandleRemovalResponse, it is possible that the Handle was successfully removed from the SRM by a previous incomplete Handle Removal Message processing. In this case, if the DRM Agent continues the Provisioning transaction with the Rights Removal in Device processing as specified in this section, then the User will lose the Rights.

The default behaviour is that the Provisioning transaction is terminated without further recovery procedures and the entry for the Provisioning transaction is removed from the Operation Log (i.e. the Rights in the source Device stay in an unusable state). A trust model may define other procedures to handle the Rights.

6.8.8 RO Removal in Device

6.8.8.1 Action Description

The DRM Agent removes the Rights from the source Device permanently. When the Rights Removal in Device processing is completed, the Provisioning transaction is terminated and the entry for the Provisioning transaction is removed from the Operation Log.

6.8.8.2 Exception Handling

There may be unexpected exceptions as specified in section 5.5.1 when removing Rights. The exception causes the removal processing to not complete.

When an exception occurs, the DRM Agent SHOULD immediately attempt to recover from it. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to the Operation Log when a new MAKE process is executed. The DRM Agent recovers from the exception by executing the Rights Removal in Device processing.

If the recovery fails, the DRM Agent MAY resume the recovery by removing the Rights from the Device when a new MAKE process is executed. To resume the recovery, the DRM Agent refers to the Operation Log as specified in section 5.5.2.

6.9 SRM Rights Upgrade

The protocols specified in this section provide necessary functions that are used for SRM Rights Upgrade.

To upgrade the Rights in SRM, the DRM Agent should request the RI to issue the upgrade rights for the SRM. As shown in Figure 17, this transaction is comprised of the OMA SCE-DRM RO Upgrade protocol and two SRM request / response message pairs: the Upgrade Rights Retrieval message pair and the Rights Upgrade message pair. The <newRO> element in the RO Upgrade Response message contains the RO with the upgraded permissions and/or constraints from the RI for the SRM.

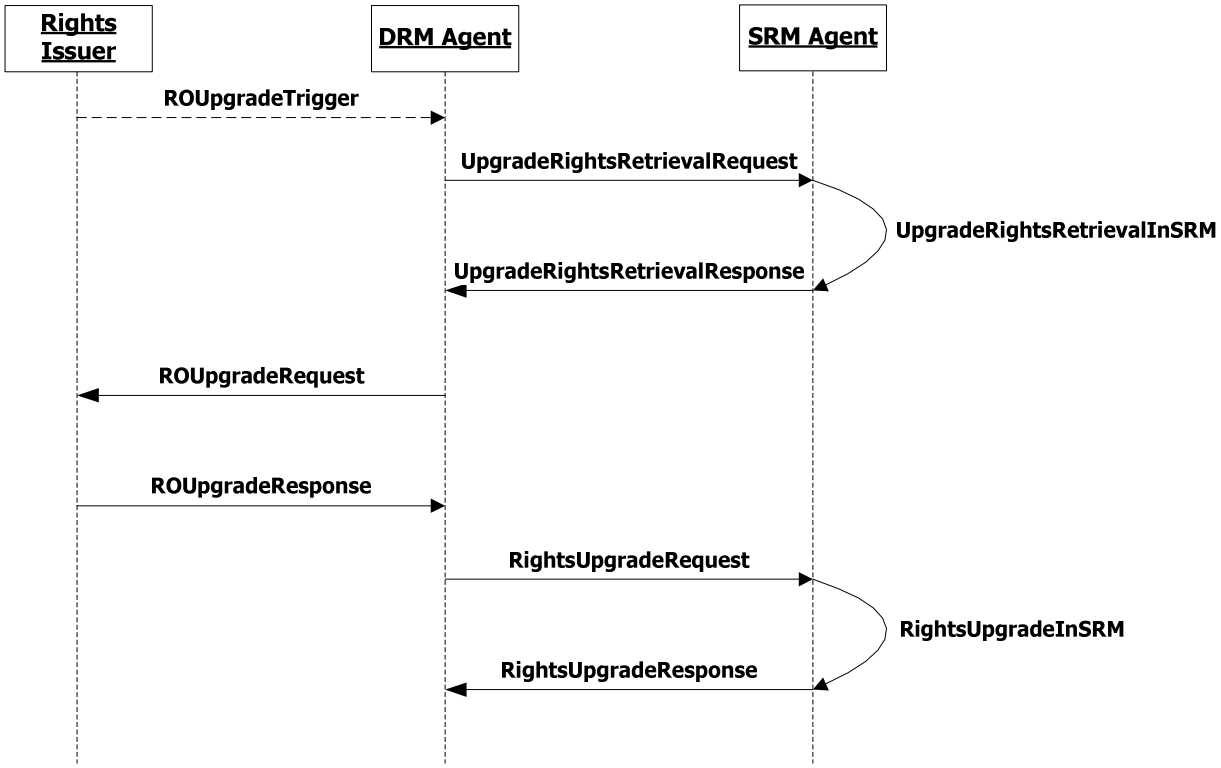

Figure 17: Sequence Diagram – SRM Rights Upgrade

6.9.1 Trigger

The SRM Rights Upgrade transaction MAY be initiated by the RO Upgrade Trigger sent from RI. The definition of RO Upgrade Trigger refers to [SCE-DRM].

In the RO Upgrade Trigger, it SHALL contain the *roID* and *assetID* element that contains the Hash Of *assetID*. The *assetID* element can be used by DRM Agent to retrieve the rights information from SRM through the protocol of "Rights Information List Query". The DRM Agent SHALL check the rightsinformation(s) contained in the "Rights Information List Query" response with the *roID* in the RO Upgrade Trigger

NOTE: The association between Handle and roID can be retrieved through the "Handle List Query" protocol,

When DRM Agent receives the RO Upgrade Trigger, DRM Agent SHALL check whether the RO specified by the *roID* is installed locally in Device or in SRM. If DRM Agent fails to find the Rights indicated by the *roID* element, DRM Agent just ignores the trigger without any further operation. If the RO is installed in SRM, DRM Agent SHALL conduct SRM Rights Upgrade transaction; else if the RO is installed in Device, DRM Agent SHALL conduct the 2-pass RO Upgrade protocol

defined in [SCE-DRM]. And if the RO Upgrade Trigger received by DRM Agent does not contain the *roID* element, DRM Agent SHALL execute the 2-pass RO Upgrade protocol defined in [SCE-DRM] as well.

The format definition of RO Upgrade Trigger is given in section 7.2.

6.9.2 Upgrade Rights Retrieval

Before the DRM Agent executes the 2-pass RO Upgrade protocol to upgrade Rights in an SRM, it SHALL retrieve the Rights from the SRM Agent through the Upgrade Rights Retrieval message pair, and then select the Rights that need to be upgraded.

The selected rights MUST be disabled (i.e. the Rights in SRM Agent stay in a disabled state. The disabled Rights cannot be used for other purposes than the current SRM Rights Upgrade transaction.)

Through the UpgradeRightsRetrievalResponse message, the DRM Agent will get the necessary information for RO Upgrade service from the "RightsInformation" field. All the Rights information received from the UpgradeRightsRetrievalResponse will be stored in local context and set in the transfer state. (The information in transfer state cannot be used or deleted for other purposes than the current SRM Rights Upgrade transaction.)

On processing the UpgradeRightsRetrieval request message, the SRM Agent SHALL reserve storage space for the upgraded rights. Since the length of the "length" element in the StateInformation () datastructure is 16 bits, the reserved storage for upgraded rights is 65536 bytes.

Generally the message format of the Upgrade Rights Retrieval protocol is the same as that of the Rights Retrieval protocol except that when the SRM fails to reserve enough storage space for the upgraded rights the Status value of Upgrade Rights Retrieval response message SHALL be set to "Storage Reservation Failed". The possible status values of UpgradeRightsRetrieval Response are defined in Table 55.

Table 55: Status of Upgrade Rights Retrieval Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Handle Not Found	The Handle in the request does not exist in the SRM.
Duplicate Handle	The SRM already has the *New Handle* and its corresponding Rights.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Storage Reservation Failed	SRM can not reserve enough storage space for the upgraded rights.
Unknown Error	Other errors

If the status value of Rights Retrieval Response is not equal to 'Success', the transaction is terminated.

For details of Rights Retrieval protocol, refer to section 6.6.1.

6.9.3 RO Upgrade

The DRM Agent executes the 2-pass RO Upgrade protocol defined in [SCE-DRM] to request upgraded Rights for Rights in an SRM. Based on the "Rights Information" field received in Rights Upgrade Retrieval session, the DRM Agent sends the RI the necessary information of the SRM Rights and the additional rights wanted via the RO Upgrade request message; the DRM Agent gets the upgraded Rights via the RO Upgrade Response message (i.e. the <newRO> element in the RO Upgrade Response message). For details of the RO Upgrade protocol, please refer to SCE DRM TS specification [SCE-DRM].

To create the RO Upgrade request message, the DRM Agent performs the following steps:

1. The DRM Agent parses the "Rights Information" field received in the RightsUpgradeRetrieval response message to create the <existingRights> element for the RO Upgrade request message.
2. The DRM Agent creates the <ROUpgradeInfo> element according to the User's desire of additional rights for the SRM Rights.

The SRM Rights Upgrade can also be initiated via the RO Upgrade Trigger issued by the RI.

In the RO Upgrade Trigger, the <roID> element MUST be equal to the RO Id that the DRM Agent gets from the SRM via parsing the Rights Information. The 'roRequested' attribute for the RO Upgrade request message SHALL be set to 'true'. The DRM Agent MUST send the whole protected RO with its state Information to RI to request the upgrade Rights for the SRM..

If there are no errors or exceptions in the RO Upgrade session (*Status* = *Success*), the DRM Agent will receive the RO that contains the upgraded permissions and/or constraints from the RI for the SRM. Especially, the RO is issued towards the SRM and the symmetric-key material encapsulated in the RO is protected under SRM's RSA public key. Then DRM Agent continues with section 6.9.4.

If the RO Upgrade operation fails for some reason (*Status* does not equal to *Success*), the DRM Agent can terminate the transaction or execute the 2-pass RO Upgrade protocol again, if the error reason of the RO Upgrade Response allows it.

6.9.4 Rights Upgrade

6.9.4.1 Description of Messages

The DRM Agent sends the RightsUpgradeRequest to the SRM Agent to upgrade the Rights in the SRM using the upgraded Rights (the <newRO> element) received in the RO Upgrade response. The fields of the request are defined in Table 56.

Table 56: Fields of RightsUpgradeRequest

Fields	Protection Requirement	Description
Handle	Integrity & Confidentiality	Same as the New Handle transmitted by the UpgradeRightsRetrievalRequest message.
Third Handle	Integrity & Confidentiality	*Third Handle* is a 10 byte random value generated by the DRM Agent for this Rights Upgrade session.
Size of Rights	Integrity	Size of the Rights in bytes. This informs the SRM Agent the size of the Rights that will be installed in the SRM. *Size of Rights* = Length of **`RightsInformation`**. .
ro element	Integrity	The part corresponds to the canonic**a**lized <ro> element **in the <protectedRO>**. It is the element that is used for the MAC value (see below).For the calculation of the MAC value, see [OMADRMv2.1].
Wrapped Key Material	Integrity	The *Wrappted Key Material* consists of a wrapped concatenation of a MAC key, KMAC and an RO encryption key, KREK. The Wrapped Key Material is same as the C value as specified in DRM v2.1 [OMADRMv2.1] section 7.2.1 while the independent random value Z is RSA encrypted by the SRM's public key.
MAC Value	Integrity	The MAC value is the same as the <mac> field in the <protectedRO> field from [OMADRMv2.1]
LAID	Integrity	Refer to section 5.1.7. This field contains the hash value of the AssetIDs that are associated with the Rights.
Rights Information	Integrity	Refer to section 5.1.6.

Before sending the RightsUpgradeRequest, the DRM Agent parses the protected RO received in the RO Upgrade response, gets the information to compose the *Rights Object Container*, *State Information*, and *REK* to be delivered to the SRM in the RightsUpgradeRequest message for SRM Rights Upgrade.

Upon receiving the RightsUpgradeRequest, the SRM Agent installs the Rights in the SRM at the storage space that is reserved in the Upgrade Rights Retrieval session. For the installation, the SRM Agent MUST perform the following procedure:

1. Verify the integrity of the request fields
2. Decrypt the *Handle* with the current Session Key
3. Compare the *Handle* with the *New Handle* in the UpgradeRightsRetrievalRequest
4. Decrypt the *Third Handle* with the current Session Key
5. Decrypt and extract the K_{MAC} and K_{REK} from the *Wrapped Key Material.*
6. Verify the MAC value on the <ro element> element using the K_{MAC} got in step 5.
7. Install the *Rights Information* and *REK* at the space reserved during the Upgrade Rights Retrieval session. Since the reserved storage space MAY be larger than the actual occupied space, the vacant part of the reserved storage SHALL be released and the occupied storage by the installation SHALL be associated with the *Third Handle*.
8. Remove the disabled Rights Information that is associated with the *Handle* from SRM

The SRM Agent sends the RightsInstallationResponse to carry the result of the procedure. The fields of the response are defined in Table 57.

Table 57: Field of RightsUpgradeResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the RightsUpgradeRequest message. The *Status* values are specified inTable 58.

Table 58: Status of Rights Upgrade Response Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by SRM Agent.
Handle Not Found	The Handle in the request does not exist in the SRM.
Handles In-consistent	The Handle in this request is different from the *Handle* in the UpgradeRightsRetrievalRequest.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
MAC verification failed	The integrity verification of the RO is not successful through verifying the MAC value.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of the fields in the response.

If no errors or exceptions occur (Status = Success), the DRM Agent MUST remove the Rights Information that was saved in the local context.

6.9.4.2 Format of Messages

The message format (**`MessageBody`**) of the RightsUpgradeRequest is specified as follows. The **`messageType`** is set to '0' and the message is protected by an HMAC.

```
roelement() {
    // Length of SignedPart element
    length                          16  uimsbf
    OctetString8()                      // Defined in Appendix E.1
}

WrappedKeyMaterial() {
    // Length of WrappedKeyMaterial element
    length                          16  uimsbf
    OctetString8()                      // Defined in Appendix E.1

}

MACValue() {
    // Length of MACValue element
    length                          16  uimsbf
    OctetString8()                      // Defined in Appendix E.1

}

MessageBody() {
    EncryptedHandle()               // Defined in Appendix E.2.4
    EncryptedThirdHandle()          //EncryptedThirdHandle() has the same format as
                                    //EncryptedHandle().  Defined in Appendix E.2.4
    Roelement()
    WrappedKeyMaterial()
    MACValue()
    sizeOfRights                    16  uimsbf
    RightsInformation()             // Defined in Appendix E.2.5.4
}
```

The fields are defined as follows:

- **`Handle`** – *Handle* field in Table 56
- **`EncryptedHandle`** – **`Handle`** encrypted with the current Session Key (SK)
- **`EncryptedThirdHandle`** – Third Handle encrypted with the current Session Key (SK)
- **`Roelement`** – The part corresponds to the canoniclaized <ro> element. It is the element that is used for the MAC value
- **`WrappedKeyMaterial`** – The Wrapped Key Material is same as the C value as specified in DRM v2.1 [OMADRMv2.1] section 7.2.1
- **`MACValue`** – The MAC value from the protected RO.
- **`sizeOfRights`** – *Size Of Rights* field in Table 56
- **`RightsInformation`** – *Rights Meta Data*, *Rights Object Container*, *State Information* fields in Table 56 (refer to Appendix E.2.5.4)

The message format (**`MessageBody`**) of the RightsUgradeResponse is specified as follows. The **`messageType`** is set to '1' and the message is protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
}
```

The field is defined as follows:

- **`Status`** - *Status* field in Table 57

6.9.4.3 Exception Handling

Refer to section 6.5.3.3.

6.10 S2S Rights Move

The protocols specified in this section provide necessary functions that are used for SRM to SRM Rights Move.

Rights in one SRM can be directly Moved to another SRM as illustrated in Figure 18. As shown in Figure 18, this transaction is comprised of four request/response message pairs: S2S Move Initiation message pair, Installation Setup message pair, Rights Installation message pair and Rights Removal message pair. If the move permission has a count constraint, the DRM Agent updates the count value in the associated StateInformation for the Moved Rights before executing the Installation Setup session.

The Installation Setup message pair SHOULD be followed by the Rights Installation message pair. If SRM Agent-2 receives any request message other than the RightsInstallationRequest following receipt of the InstallationSetupRequest, SRM Agent-2 SHOULD return Unexpected Request in the Status field of the response message.

In this section, the session key between SRM Agent-1 and the DRM Agent is called S1 session key (SK1), and the session key between SRM Agent-2 and the DRM Agent, S2 session key (SK2).

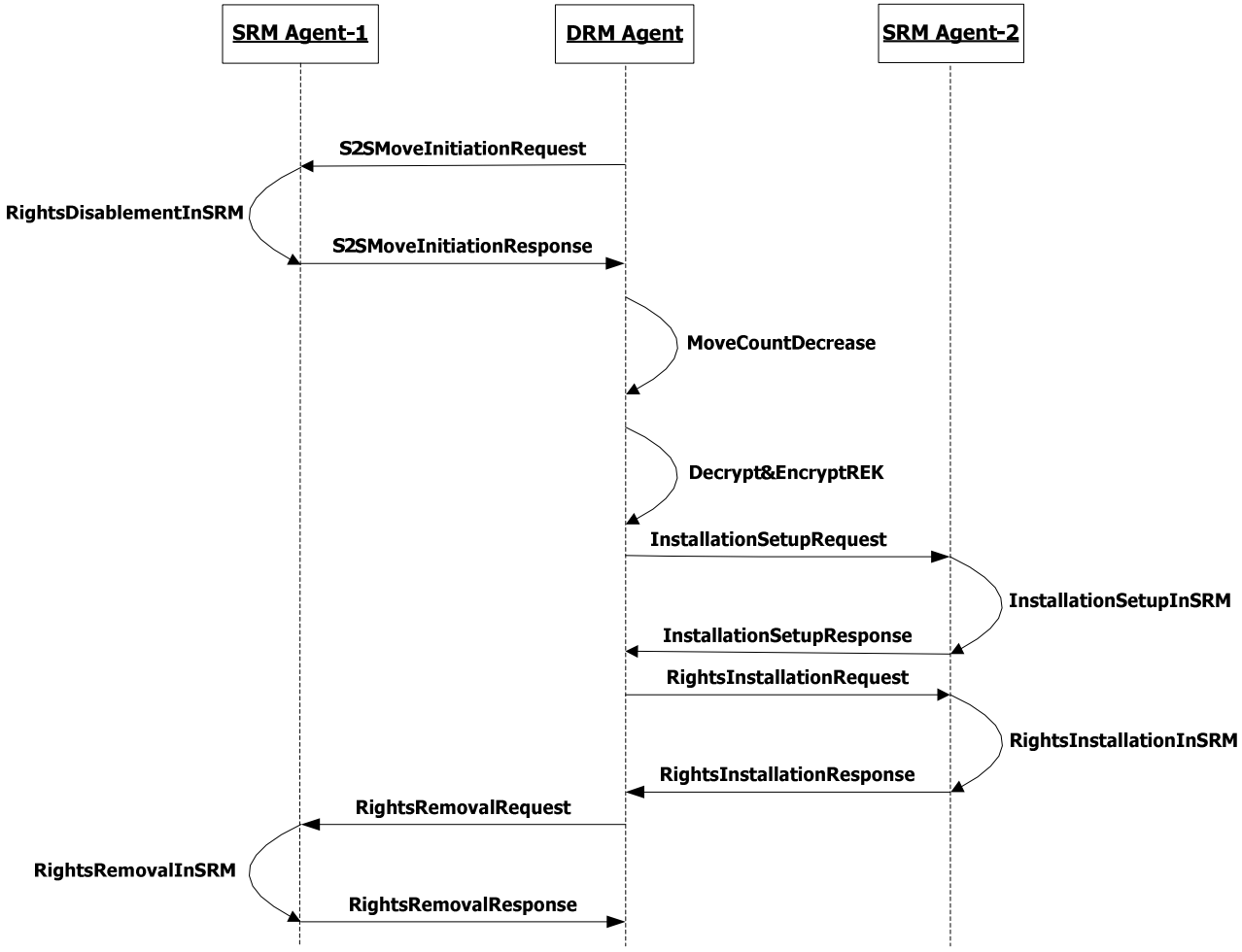

Figure 18: Sequence Diagram – SRM to SRM Move

6.10.1 S2S Move Initiation

6.10.1.1 Description of Messages

The DRM Agent sends the S2SMoveInitiationRequest to initiate the Move of Rights from the source SRM (SRM Agent-1) to the target SRM (SRM Agent-2).

The fields of the S2SMoveInitiationRequest are defined in Table 59.

Table 59: Fields of S2SmoveInitiationRequest

Fields	Protection Requirement	Description
Target SRM Id	Integrity	*Target SRM Id* contains the SRM ID of SRM Agent-2 in the S2S Move transaction.
Handle	Integrity	This identifies Rights that will be Moved from SRM Agent-1 to SRM Agent-2. Refer to section 5.1.3.
New Handle	Integrity & Confidentiality	*New Handle* is a 10 byte random value generated by the DRM Agent for this Move transaction.

Upon receiving the S2SMoveInitiationRequest , SRM Agent-1 MUST perform the following procedure:

1. Verify the integrity of the request fields
2. Verify the secure status of SRM Agent-2 through CRL
3. If SRM Agent-2 is reliable, find the Rights corresponding to *Handle*
4. If found, then decrypt the *New Handle* with the S1 Session Key
5. Check if the SRM already has the same Handle with *New Handle*. If yes, the SRM Agent sets *Status* to *Duplicate Handle*. If no, overwrite the *Handle* in the SRM with the *New Handle*, and disable the Rights. The disabled Rights cannot be used for the other purposes except the current S2S Move transaction.

The SRM Agent-1 sends the S2SMoveInitiationResponse to carry the result of the procedure. The fields of the response are defined in Table 60.

Table 60: Fields of S2SMoveInitiationResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the S2SMoveInitiationRequest message. The *Status* values are specified in Table 61. If *Status* contains any error, only this field is present in the S2SMoveInitiationResponse.
Rights Information	Integrity	Refer to section 5.1.6.
REK	Integrity & Confidentiality	Encrypted with the S1 Session Key. Refer to section 5.1.1.4.

Table 61: Status of S2SMoveInitiation Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by SRM Agent-1.
Handle Not Found	The Handle in the request does not exist in SRM Agent-1.
Target SRM not Accepted	The Target SRM indicated by the Target SRM Id is not reliable or does not exist.
Duplicate Handle	SRM Agent-1 already has the *New Handle* and its corresponding Rights.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent MUST perform the following procedure:

1. Verify the integrity of fields in the response
2. Save the information received from the S2SMoveInitiationResponse in local context and set them in the transfer state. The information in transfer state cannot be used or deleted for the other purposes except the current S2S Move transaction.)

If no errors or exceptions (*Status* = *Success*), the DRM Agent continues with section 6.10.2.

6.10.1.2 Format of Messages

The message format (**MessageBody**) of the S2SMoveInitiationRequest is specified as follows. The **messageType** is set to '0' and the message is protected by an HMAC.

```
TargetSRMId(){
    EntityId()                      // Defined in Section 6.1.1.2
}


MessageBody() {
    TargetSRMID()
    Handle()                        // Defined in Appendix E.2.4
    EncryptedNewHandle()            // Defined in Appendix E.2.9
}
```

The fields are defined as follows:

- **TargetSRMID**– *Target SRM ID* field in Table 59
- **Handle** –*Handle* field in Table 59
- **EncryptedNewHandle** – *New Handle* field in Table 59 encrypted with the current Session Key (SK1)

The message format (**MessageBody**) of the S2SMoveInitiationResponse is specified as follows. The **messageType** is set to '1' and the message is protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        RightsInformation()         // Defined in Appendix E.2.5.4
        EncryptedRek()              // Defined in Appendix E.2.7

    }
}
```

The fields are defined as follows:

- **Status** - *Status* field in Table 60
- **RightsInformation** – *Rights Meta Data*, *Rights Object Container*, *State Information* fields in Table 60 (Refer to Appendix E.2.5.4)
- **EncryptedRek** – *REK* field in Table 60 (**Rek** in Appendix E.2.5.5) encrypted with the current Session Key (SK1)

6.10.1.3 Exception Handling

Refer to section 6.6.1.3 except that the message name is different to its counter part in section 6.6.1.3.

6.10.2 Move Count Decrease

6.10.2.1 Action Description

The DRM Agent performs the following procedure:

1. Verify whether there is a count constraint for the Move permission. If not, proceed to step 4.
2. Check whether the count constraint for the Move permission is less than 1. If not, decrease the value of Move count in StateInformation by 1.
3. If the Move count equals 0, remove the move permission from the StateInformation field, else update the value of the Move count in the StateInformation field with the new value from step 2.
4. Check whether SRM Agent-2 is still connected. If not, terminate the S2S Move transaction.

After updating the "StateInformation" field and confirming that SRM Agent-2 is connected, the DRM Agent continues with section 6.10.3.

6.10.3 Decrypt & Encrypt REK

6.10.3.1 Action Description

The DRM Agent decrypts the EncryptedRek from the S2SMoveInitiationResponse message with the S1 session key and then encrypts the REK with the S2 session key creating a new EncyptedRek that will be sent in the RightsInstallationRequest message.

After the new EncryptedRek is produced, the DRM Agent SHALL delete the plain REK immediately and continue with section 6.10.4.

6.10.4 Installation Setup

6.10.4.1 Description of Messages

The DRM Agent sends the InstallationSetupRequest to verify whether SRM Agent-2 has enough space to install the Rights. The fields of the request are defined in Table 34.

Upon receiving the InstallationSetupRequest, the SRM Agent MUST perform the following procedure:

1. Verify the integrity of the fields
2. Decrypt the *Handle* with the S2 Session Key (SK2)
3. Check if it already has the same Handle. If yes, the SRM Agent sets *Status* to *Duplicate Handle* and sends the InstallationSetupResponse as described below.
4. Checks if the SRM has space for the new Rights. If not, the SRM Agent sets *Status* to *Not Enough Space*. Otherwise, it stores the *Handle* in the SRM securely. The *Handle* is not included in the Handle List until the Move transaction is completed.

The SRM Agent sends the InstallationSetupResponse to carry the result of the procedure. The fields of the response are defined in Table 35.

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response.

If no errors or exceptions occur (*Status* = *Success*), the DRM Agent continues with section 6.10.5

6.10.4.2 Format of Messages

The message format (**`MessageBody`**) of the InstallationSetupRequest and InstallationSetupResponse refers to section 6.5.1.2.

6.10.4.3 Exception Handling

Refer to section 6.5.1.3 except that when this transaction will be terminated, the DRM Agent sends the RightsEnablementRequest as specified in section 6.12.5 in order to enable the Rights. The *New Handle* used in S2SmoveInitiationRequest MUST be used in this request. If the RightsEnablementResponse contains a *Status* of either *Success* or *Handle Not Found*, then this transaction is terminated.

6.10.5 Rights Installation

6.10.5.1 Description of Messages

The DRM Agent sends the RightsInstallationRequest to install the Rights in SRM Agent-2. The fields of the request are defined in Table 62.

Table 62: Fields of RightsInstallationRequest

Fields	Protection Requirement	Description
Handle	Integrity & Confidentiality	Same as the Handle transmitted by the InstallationSetupRequest in Table 34.
REK	Integrity & Confidentiality	Encrypted with the S2 Session Key. Refer to section 5.1.1.4.
LAID	Integrity	Refer to 5.1.7. This contains the hash value of the AssetIDs that are associated with the Rights.
Rights Information	Integrity	Refer to section 5.1.6.

Upon receiving the RightsInstallationRequest, SRM Agent-2 installs the Rights in the SRM. For the installation, SRM Agent-2 MUST perform the following procedure:

1. Verify the integrity of the request fields
2. Decrypt the *Handle* and *REK* with the S2 Session Key(SK2)
3. Compare the *Handle* with the *Handle* in the InstallationSetupRequest
4. Install the *Rights Information* and *REK* at a space associated with the *Handle*.

The SRM Agent sends the RightsInstallationResponse to carry the result of the procedure. The fields of the response are defined in Table 63.

Table 63: Fields of RightsInstallationResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the RightsInstallationRequest message. The *Status* values are specified in Table 64.

Table 64: Status of Rights Installation Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by SRM Agent-2.
Handle Not Found	The Handle in the request does not exist in SRM Agent-2.

Status Value	Description
Handles In-consistent	The Handle in this request is different from the *Handle* in the InstallationSetupRequest.
Not Enough Space	The size of *Rights Information* exceeds *Size of Rights* in Table 62.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response.

If no errors or exceptions (*Status* = *Success*), the DRM Agent continues with section 6.10.6.

6.10.5.2 Format of Messages

The message format (**`MessageBody`**) of the RightsInstallationRequest is specified as follows. The **`messageType`** is set to '0' and the message is protected by an HMAC.

```
EncryptedHandle() {
    // Contains the encrypted Handle and REK
    EncryptedData()                // Defined in Appendix E.1
}

MessageBody() {
    EncryptedHandle()
    EncryptedRek()                 // Defined in Appendix E.2.5.5
    Laid()                         // Defined in Appendix E.3
    RightsInformation()            // Defined in Appendix E.2.5.4
}
```

The fields are defined as follows:

- **`Handle`** – *Handle* fields in Table 62
- **`EncryptedHandle`** – Encrypted **`Handle`** with the current Session Key (SK2)
- **`EncryptedRek`**– Encrypted **`REK`** with the S2 Session Key (SK2)
- **`Laid`** – *LAID* field in Table 62
- **`RightsInformation`** – *Rights Meta Data*, *Rights Object Container*, *State Information* fields in Table 62 (Refer to Appendix E.2.5.4)

The message format (**`MessageBody`**) of the RightsInstallationResponse is specified as follows. The **`messageType`** is set to '1' and the message is protected by an HMAC.

```
MessageBody() {
    Status()                       // Defined in Appendix E.2.2
}
```

The field is defined as follows:

- **`Status`** - *Status* field in Table 63

6.10.5.3 Exception Handling

Refer to section 6.5.1.3 except that the message name is different to its counterpart in section 6.5.1.3.

Besides, when this transaction will be terminated, the DRM Agent sends the RightsEnablementRequest as specified in section 6.12.5 in order to enable the Rights. The *New Handle* used in S2SmoveInitiationRequest MUST be used in this request. If the RightsEnablementResponse contains a *Status* of either *Success* or *Handle Not Found*, then this transaction is terminated.

6.10.6 Rights Removal

6.10.6.1 Description of Messages

The DRM Agent executes the Rights Removal Message processing as specified in section 6.12.6 in order to remove the original Rights from SRM Agent-1.

The *Handle* in the RightsRemovalRequest MUST be identical to the *New Handle* in the previous S2SMoveInitiation specified in section 6.10.1.

When the Rights Removal Message processing is completed (*Status* = *Success*), the S2S Move is terminated and the entry for the S2S Move transaction is removed from the Operation Log.

The DRM Agent also removes from local context the information received in S2SMoveInitiation response message in this S2S Move transaction.

6.10.6.2 Format of Messages

Refer to section 6.12.6.2.

6.10.6.3 Exception Handling

Refer to section 6.12.6.3.

6.11 SRM extensions for BCAST service support

6.11.1 Movement of BCAST Tokens from Device to SRM

BCAST Tokens are moved from a Device to an SRM via the Device to SRM BCAST Token Move transaction as illustrated in Figure 19. The transaction is comprised of a single request-response pair: the BCASTTokenInstallationRequest conveys the BCAST Token Container (specified in section 5.1.10) to be installed in the SRM; the BCASTTokenInstallationResponse contains the status of the BCAST Token Installation in the SRM.

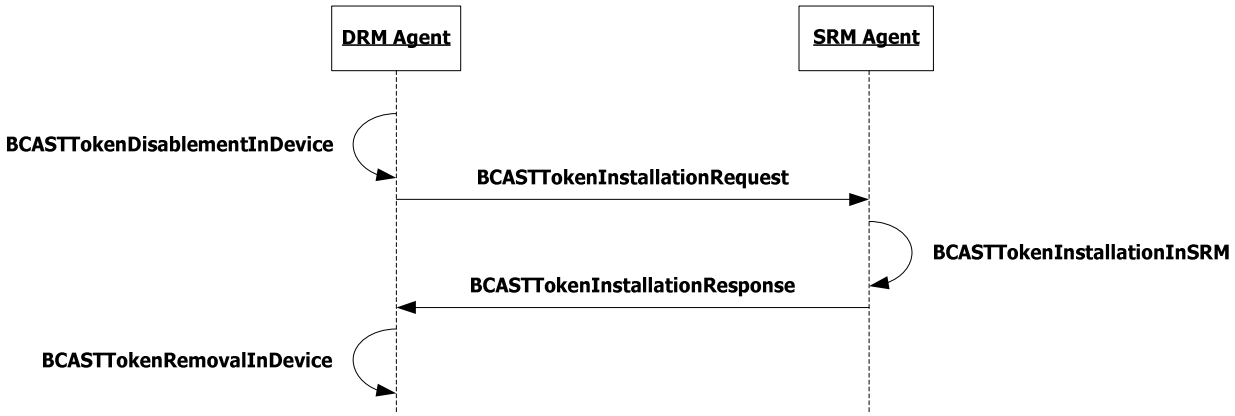

Figure 19: Sequence Diagram – BCAST Token Move from a Device to SRM

The BCAST Token Move from a Device to SRM transaction is defined as follows:

1. Before sending BCASTTokenInstallationRequest, the DRM Agent SHALL perform the following actions:
    a. Check if *Movable* attribute defined in section 5.1.10 allows BCAST Token Move for the selected BCAST Token. The *Movable* attribute SHALL be processed as defined in [DRMXBSv1.1].
    b. If the BCAST Tokens can be moved then perform BCAST Token Disablement in the Device as defined in section 6.11.1.1. Otherwise the DRM Agent MUST NOT perform BCAST Token Move.
2. The DRM Agent sends a BCASTTokenInstallationRequest to the SRM Agent.
3. Upon reception of the BCASTTokenInstallationRequest, the SRM Agent performs BCAST Token Installation in the SRM.
4. The SRM Agent sends BCASTTokenInstallationResponse to the DRM Agent.
5. Upon reception of BCASTTokenInstallationResponse, the DRM Agent performs BCAST Token Removal in the Device.

6.11.1.1 BCAST Token Disablement in the Device

6.11.1.1.1 Action Description

The DRM Agent disables the BCAST Tokens. The disabled BCAST Tokens cannot be used for any purposes other than the current BCAST Token Move transaction. After disabling the BCAST Tokens, the DRM Agent SHALL continue with section 6.11.1.2 (BCAST Token Installation in the SRM).

6.11.1.1.2 Exception Handling

If the BCAST Token cannot be disabled, the Device SHALL abandon BCAST Tokens Move.

6.11.1.2 BCAST Token Installation in the SRM

6.11.1.2.1 Description of Messages

The DRM Agent sends BCASTTokenInstallationRequest to install the BCAST Tokens in the SRM. The fields of the request are defined in Table 65.

Table 65: Fields of BCASTTokenInstallationRequest

Fields	Protection Requirement	Description
BCAST TokenContainer	Integrity	Refer to section 5.1.10

Upon receiving the BCASTTokenInstallationRequest, the SRM Agent SHALL perform the following actions:

1. Verify the integrity of the fields in the BCASTTokenInstallationRequest.
2. Install the BCAST Tokens and related attributes of from the BCAST Token Container of the BCASTTokenInstallationRequest in the SRM.

The SRM Agent sends the RightsInstallationResponse to carry the result of the procedure. The fields of the response are defined in Table 66.

Table 66: Fields of BCASTTokenInstallationResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the BCASTTokenInstallationRequest. The Status values are specified in Table 67.

Table 67: Status of BCAST Token Installation Response Message

Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Not Enough Space	There is not enough space to store the transferred BCAST Tokens.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response. If no errors (*Status* = *Success*) or exceptions occur, the DRM Agent SHALL perform BCAST Token Removal in the Device as defined in section 6.11.1.3.

In the case value of the *Status* field in BCASTTokenInstallationResponse message processing differs from "Success", the DRM Agent MAY generate new BCASTTokenInstallationRequest. If BCAST Token Installation in the SRM cannot be completed successfully, the DRM Agent SHALL perform BCAST Token Enablement in the Device and abandon BCAST Token Move Transaction.

6.11.1.2.2 Format of Messages

The message format (**`MessageBody`**) of the BCASTTokenInstallationRequest is specified as follows. The **`messageType`** is set to '0' and the message is protected by an HMAC.

```
MessageBody() {
    BCASTTokenContainer()          // Defined in Appendix E.2.10.7
}
```

The message format (**MessageBody**) of the RightsInstallationResponse is specified as follows. The **messageType** is set to '1' and the message is protected by an HMAC.

```
MessageBody() {
    Status()                       // Defined in Appendix E.2.2
}
```

6.11.1.2.3 Exception Handling

There may be unexpected exceptions during the BCAST Token Installation in the SRM. The DRM SHOULD attempt to recover from the exception. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to Operation Log when a new MAKE process is executed.

[**Recovery Procedure –BCAST Token Move Cancellation]**

The DRM Agent SHOULD attempt to cancel BCAST Token Move transaction. In that case, the DRM Agent SHALL use BCAST Token Information Retrieval procedure to check if the BCAST Tokens were successfully installed in the SRM.

- If the BCASTTokens are not found on the SRM, the DRM Agent MUST enable the BCAST Tokens on the Device.
- Otherwise, the DRM Agent MUST perform BCAST Token Removal in the SRM and re-enable BCAST Tokens on the Device.

6.11.1.3 BCAST Token Removal in the Device

6.11.1.3.1 Action Description

The DRM Agent removes the BCAST Tokens from the source Device permanently.

6.11.1.3.2 Exception Handling

There may be unexpected exceptions during BCAST Token Removal in Device. The DRM SHOULD attempt to recover from the exception. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to Operation Log when a new MAKE process is executed.

[**Recovery Procedure – BCAST Token Move Cancellation]**

The DRM Agent SHOULD attempt to cancel BCAST Token Move transaction. In that case, the DRM Agent MUST perform BCAST Token Removal in the SRM and re-enable the BCAST Token on the Device.

6.11.2 Movement of BCAST Tokens from SRM to Device

BCAST Tokens are moved from an SRM to Device via SRM to Device BCAST Token Move transaction as illustrated in Figure 20. It is comprised of two protocols: BCAST Token Retrieval from the SRM and BCAST Token Removal from the SRM.

The BCAST Token Retrieval Protocol consists of a message pair: BCASTTokenRetrievalRequest identifies the BCAST Tokens to be Moved; BCASTTokenRetrievalResponse contains the BCASTTokens (in a BCAST Token Container) and the status of BCASTTokenRetrievalRequest processing.

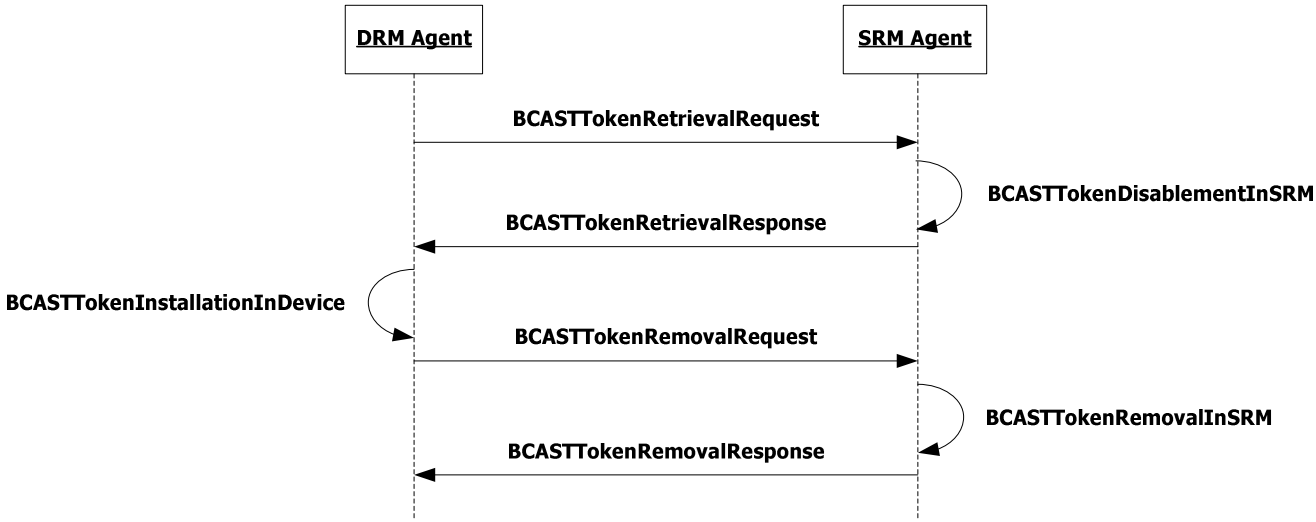

Figure 20: Sequence Diagram –BCAST Token Move from SRM to Device

Token Move from a SRM to Device transaction is defined as follows:

1. The DRM Agent sends BCASTTokenRetrievalRequest to the SRM Agent.
2. Upon reception of TokenRetrievalRequest, the SRM Agent SHALL perform the following actions:
    a. Locate the BCAST Tokens on the SRM.
    b. Disable the BCAST Tokens on the SRM (see section 6.11.2.2).
3. The SRM Agent sends BCASTTokenRetrievalResponse to the DRM Agent.
4. Upon reception of TokenRetrievalResponse, the DRM Agent SHALL perform the following actions:
    a. Check if *Movable* attribute defined in section 5.1.10 allows BCAST Token Move for retrieved the BCAST Tokens.
    b. If the BCAST Tokens can be moved then initiate the BCAST Token Removal from the SRM protocol (as defined in section 6.11.5). Otherwise restore the BCAST Token Move transaction as defined in section 5.5.3.
5. If BCAST Token Removal in the SRM protocol cannot be completed successfully (i.e. unexpected exceptions occur), the DRM Agent SHALL rollback BCAST Token Move transaction as defined in section 5.5.3.

6.11.2.1 BCAST Token Retrieval from the SRM

6.11.2.1.1 Description of Messages

The DRM Agent sends BCASTTokenRetrievalRequest to the SRM Agent to request BCAST Token Move from SRM to Device. The fields of the request are defined in Table 68.

Table 68: Fields of BCASTTokenRetrievalRequest

Fields	Protection Requirement	Description
RI ID	Integrity	Refer to section 5.1.10
Token Delivery ID	Integrity	Refer to section 5.1.10

Upon receiving the BCASTTokenRetrievalRequest, the SRM Agent SHALL perform the following actions:

1. Verify the integrity of the fields in BCASTTokenRetrievalRequest.
2. Locate the BCAST Tokens on the SRM using RI ID and Token Delivery ID from BCASTTokenRetrievalRequest.
3. Disable the BCAST Tokens on the SRM as defined in section 6.11.2.2.

The SRM Agent sends the BCASTTokenRetrievalResponse to carry the result of the procedure. The fields of the response are defined in Table 69.

Table 69: Fields of BCASTTokenRetrievalResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the BCASTTokenRetrievalRequest. The Status values are specified in Table 70.
BCAST Token Container	Integrity	Refer to section 5.1.10

Table 70: Values of *Status* field of the BCASTTokenRetrievalResponse

Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Token Not Found	The BCAST Tokens are not found on SRM.
Disablement Failed	The SRM Agent failed to disable the BCAST Tokens on SRM.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response. If no errors (*Status* = *Success*) or exceptions occur, the DRM Agent SHALL perform BCAST Token Installation in the Device.

In the case value of the *Status* field in BCASTTokenInstallationResponse message processing differs from “Success”, the DRM Agent MAY generate new BCASTTokenInstallationRequest.

6.11.2.1.2 Format of Messages

The message format (**`MessageBody`**) of the BCASTTokenRetrievalRequest is specified as follows. The **`messageType`** is set to ‘0’ and the message is protected by an HMAC.

```
MessageBody() {
    RiId()                          // Defined in Appendix E.2.5
    TokenDeliveryId()               // Defined in Appendix E.2.10.4
}
```

The message format (**`MessageBody`**) of the BCASTTokenRetrievalResponse is specified as follows. The **`messageType`** is set to ‘1’ and the message is protected by an HMAC.

```
MessageBody() {
    Status()                          // Defined in Appendix E.2.2
    if ( Status == 0 ) {
         BCASTTokenContainer()        // Defined in Appendix E.2.10.7
     }
}
```

6.11.2.1.3 Exception Handling

There may be unexpected exceptions during the BCAST Token Retrieval messages processing. The DRM SHOULD attempt to recover from the exception. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to Operation Log when a new MAKE process is executed.

[**Recovery Procedure –BCAST Token Move Cancellation]**

The DRM Agent SHOULD attempt to cancel BCAST Token Move transaction. In that case, the DRM Agent MUST re-enable the associated BCAST Tokens in the SRM (if disabled) using BCAST Token Enablement procedure specified in section 6.11.3.3 and abandon BCAST Token Move transaction.

6.11.2.2 BCAST Token Disablement in the SRM

6.11.2.2.1 Action Description

The SRM Agent disables the BCAST Tokens. The disabled amount of BCAST Token SHALL be stored in the *Token Quantity in Use* attribute in the SRM and cannot be used for the other purposes except the current BCAST Token Move from SRM to Device transaction. The disabled BCAST Tokens can only be enabled by the Device which induced BCAST Token disablement.

6.11.2.2.2 Exception Handling

If BCAST Token cannot be disabled, the SRM Agent SHALL generate relevant error code.

6.11.2.3 BCAST Token Installation in the Device

6.11.2.3.1 Action Description

The DRM Agent installs the BCAST Tokens in the Device. Upon completing of BCAST Token Installation in the Device, the DRM Agent SHALL perform BCAST Token Removal from the SRM.

6.11.2.3.2 Exception Handling

There may be unexpected exceptions during the BCAST Token Installation in the Device or BCAST Token Removal from the SRM. The DRM SHOULD attempt to recover from the exception. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to Operation Log when a new MAKE process is executed.

[**Recovery Procedure –BCAST Token Move Cancellation]**

The DRM Agent SHOULD attempt to cancel the BCAST Token Move transaction. In that case, the DRM Agent MUST re-enable the BCAST Tokens in the SRM using BCAST Token Enablement procedure specified in section 6.11.3.3 and abandon BCAST Token Move transaction.

6.11.3 Local BCAST Token Consumption by the Device

This section defines a mechanism which allows a Device to consume BCAST Tokens that are stored on SRM without Moving BCAST Tokens to a Device (e.g. BCAST Token Move transaction may be prohibited by *Movable* attribute).

Local BCAST Token Consumption transaction is comprised of two phases:

1. Retrieval of selected BCAST Tokens from the SRM and BCAST Token consumption by the Device:
   - BCASTTokenConsumptionRequest identifies the BCAST Token to be transferred from SRM to Device
   - BCASTTokenConsumptionResponse transfers the BCAST Token to Device
2. Update of BCAST Token data and BCAST Token Enablement in SRM
   - BCASTTokenEnablementRequest transfers the updated BCAST Token Container to the SRM Agent
   - BCASTTokenEnablementResponse conveys status of BCAST Token update and enablement in SRM

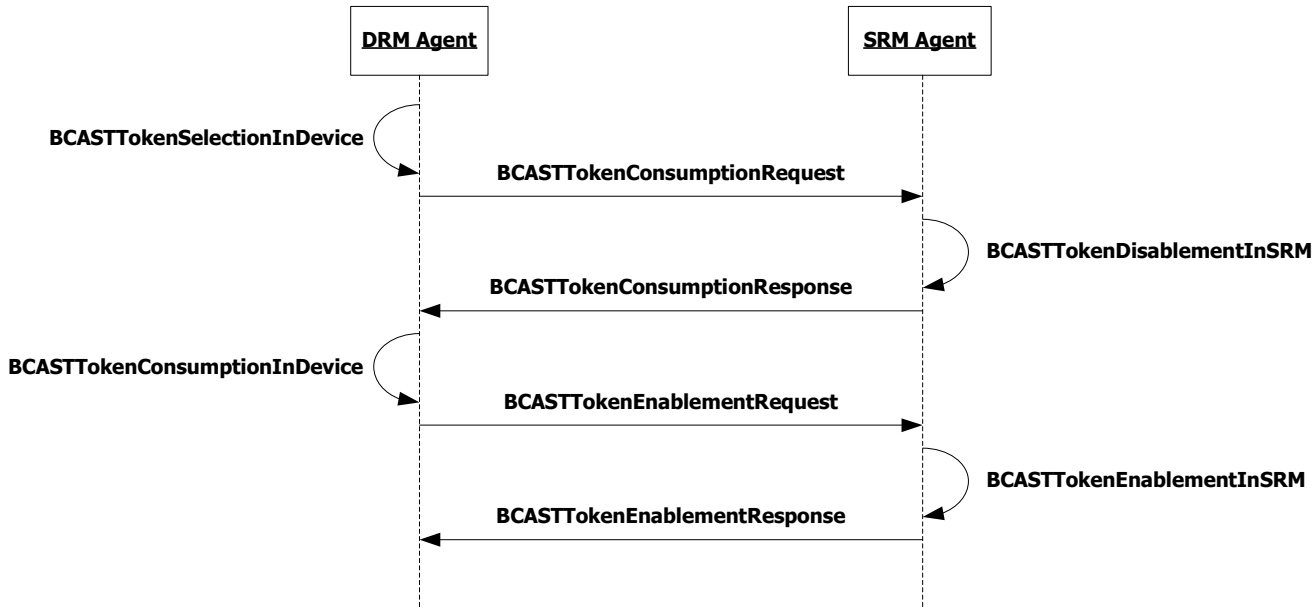

Figure 21: Sequence Diagram – Local BCAST Token Consumption

Local BCAST Token Consumption transaction is defined as follows:

1. The DRM Agent selects BCAST Token for consumption. If information about the BCAST Tokens stored on the SRM is not available on the device, it SHALL be retrieved from the SRM using BCAST Token Information Retrieval protocol (refer to section 6.11.4.1).
2. The DRM Agent sends BCASTTokenConsumptionRequest to the SRM Agent.
3. Before sending BCASTTokenConsumptionResponse, the SRM Agent SHALL perform BCAST Token Disablement in SRM.
4. The SRM Agent sends BCASTTokenConsumptionResponse.
5. BCAST Token is used by the DRM Agent for BCAST Token-based contents consumption.
6. After BCAST Token consumption is completed, the DRM Agent sends a BCASTTokenEnablementRequest to the SRM Agent.

7. Upon reception of BCASTTokenEnablementRequest, the SRM Agent updates BCAST Token attributes stored on the SRM and performs BCAST Token Enablement in the SRM.
8. The SRM Agent sends BCASTTokenEnablementResponse.

6.11.3.1 BCAST Token Consumption Request

6.11.3.1.1 Description of Messages

The DRM Agent sends BCASTTokenConsumptionRequest to the SRM Agent to request BCAST Token for consumption by the Device. The fields of the request are defined in Table 71.

Table 71: Fields of BCASTTokenConsumptionRequest

Fields	Protection Requirement	Description
RI ID	Integrity	Refer to section 5.1.10
Token Delivery ID	Integrity	Refer to section 5.1.10
Requested Amount of BCAST Tokens	Integrity	Contains the requested amount of BCAST Tokens.

Upon receiving the BCASTTokenConsumptionRequest, the SRM Agent SHALL perform the following actions:

1. Verify the integrity of the fields in BCASTTokenConsumptionRequest.
2. Locate BCAST Token on the SRM using RI ID and Token Delivery ID.
3. Check if stored value of *Token Quantity* attribute is more than or equals value of the *Requested Amount of BCAST Token* in the BCASTTokenConsumptionRequest.
4. If *Requested Amount of BCAST Tokens* successfully validates against *Token Quantity* value stored on the SRM, the SRM Agent SHALL disable the requested amount of BCAST Tokens on the SRM as defined in section 6.11.3.2.

The SRM Agent sends the BCASTTokenConsumptionResponse carrying the result of the procedure to the DRM Agent. The fields of the response are defined in Table 72.

Table 72: Fields of TokenConsumptionResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the BCASTTokenConsumptionRequest. The *Status* values are specified in Table 73.
Service ID/Program IDs	Integrity	Refer to section 5.1.10
Domain IDs	Integrity	Refer to section 5.1.10
Movable	Integrity	Refer to section 5.1.10
Reporting Information	Integrity	Refer to section 5.1.10
Token Quantity	Integrity	Granted amount of BCAST Tokens which equals to *Requested Amount of BCAST Tokens* in the BCASTTokenConsumptionRequest.

Table 73: Values of *Status* field of the BCASTTokenConsumptionResponse

Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Token Not Found	The Token Delivery ID is not found on SRM.

Value	Description
Insufficient Amount of BCAST Tokens	Value of the stored *Token Quantity* parameter for requested BCAST Tokens is less than *Requested Amount of BCAST Token* in the BCATToken ConsumptionRequest.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response. If no errors or exceptions occur (*Status* = *Success*), the Device can consume the BCAST Tokens. After BCAST Token consumption is completed, the Device SHALL perform BCAST Token Enablement in the SRM.

In the case value of the *Status* field in BCASTTokenConsumptionResponse message processing differs from "Success", the DRM Agent MAY send another BCASTTokenConsumptionRequest or abandon Local BCAST Token Consumption transaction.

6.11.3.1.2 Format of Messages

The message format (**`MessageBody`**) of the BCASTTokenConsumptionRequest is specified as follows. The **`messageType`** is set to '0' and the message is protected by an HMAC.

```
MessageBody() {
   RiId()                          // Defined in Appendix E.2.5
   TokenDeliveryId()               // Defined in Appendix E.2.10.4
   TokenQuantity()                 // Defined in Appendix E.2.10.6
}
```

The fields are defined as follows:

- **`TokenQuantity`** – *Requested Amount of BCAST Token* field in Table 71.

The message format (**`MessageBody`**) of the BCASTTokenConsumptionResponse is specified as follows. The **`messageType`** is set to '1' and the message is protected by an HMAC.

```
MessageBody() {
   Status()                        // Defined in Appendix E.2.2
   if( Status == 0 ) {
      ServiceProgramIds()          // Defined in Appendix E.2.10.1
      Movable()                    // Defined in Appendix E.2.10.2
      DomainIds()                  // Defined in Appendix E.2.10.3
      ReportingInformation()       // Defined in Appendix E.2.10.5
      TokenQuantity()              // Defined in Appendix E.2.10.6
   }
}
```

6.11.3.1.3 Exception Handling

There may be unexpected exceptions during the BCAST Token Consumption messages processing. The DRM Agent SHOULD attempt to recover from the exception. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to Operation Log when a new MAKE process is executed.

[**Recovery Procedure –BCAST Token Move Cancellation]**

The DRM Agent SHOULD attempt to cancel Local BCAST Token Consumption transaction. In that case, the DRM Agent MUST re-enable BCAST Token in the SRM (if disabled) using BCAST Token Enablement procedure specified in section 6.11.3.3 and abandon Local BCAST Token Consumption transaction.

6.11.3.2 Disablement of BCAST Tokens in the SRM

6.11.3.2.1 Action Description

The SRM Agent disables the amount of BCAST Tokens which is currently in use by Local BCAST Token Consumption transaction. This amount SHALL be stored in the *Token Quantity in Use* attribute, and cannot be used for the purposes other than the current Local BCAST Token Consumption transaction. The disabled amount of BCAST Tokens SHALL be subtracted from the current Token Quantity. The disabled amount of BCAST Tokens MUST only be enabled during transaction with the Device which induced BCAST Token disablement.

The Token Quantity that remains enabled on the SRM MAY be used by another DRM.

6.11.3.2.2 Exception Handling

If BCAST Tokens cannot be disabled, the SRM Agent SHALL generate relevant error code.

6.11.3.3 BCAST Token Enablement in the SRM

6.11.3.3.1 Description of Messages

The DRM Agent sends BCASTTokenEnablementRequest to the SRM Agent to enable BCAST Token in the SRM. The fields of the request are defined in Table 74.

Table 74: Fields of BCASTTokenEnablementRequest

Fields	Protection Requirement	Description
RI ID	Integrity	Refer to section 5.1.10
Token Delivery ID	Integrity	Refer to section 5.1.10
Reporting Information	Integrity	Refer to section 5.1.10. This field is optional. It SHALL be included if *Reporting Information* in the Device differs from the *Reporting Information* in the SRM.
Remaining Amount of BCAST Token	Integrity	Contains the amount of BCAST Token remaining after local BCAST Token consumption or the amount of BCAST Tokens that remains disabled in the SRM after unsuccessful BCAST Token Move from SRM to Device transaction.

Upon receiving the BCASTTokenConsumptionRequest, the SRM Agent SHALL perform the following actions:

1. Verify the integrity of the fields in BCASTTokenRetrievalRequest.
2. Locate BCAST Token in the SRM using RI ID and Token Delivery ID.
3. Check if value of the *Token Quantity in Use* attribute for the current transaction is more than or equals value of the *Remaining Amount of BCAST Token* field in the BCASTTokenEnablementRequest. If yes, the SRM Agent SHALL add the value of the Remaining Amount of BCAST Token to the stored value of the *Token Quantity*, clear *Token Quantity in Use* for the current transaction and update *Reporting Information* (if present in the request) in the SRM. Otherwise the SRM Agent SHALL terminate BCASTTokenEnablementRequest processing and generate BCASTTokenEnablementResponse indicating “Invalid BCAST Token Amount” in the *Status* field.

The SRM Agent sends the BCAST TokenEnablementResponse to carry the result of the procedure. The fields of the response are defined in Table 75.

Table 75: Fields of BCASTTokenEnablementResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the BCASTTokenEnablementRequest. The *Status* values are specified in Table 76.

Table 76: Values of *Status* field of the BCASTTokenEnablementResponse

Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Token Not Found	The Token Delivery ID is not found on SRM.
Invalid BCAST Token Amount	Remaining Amount of BCAST Tokens in the BCASTTokenEnablementRequest does not appear to be valid.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response. If no errors (*Status* = *Success*) or exceptions occur, the current transaction is completed.

In the case value of the *Status* field in BCASTTokenEnablementResponse message processing differs from "Success", the DRM Agent MAY send another BCASTTokenEnablementRequest or abandon the current transaction.

6.11.3.3.2 Format of Messages

The message format (**`MessageBody`**) of the BCASTTokenEnablementRequest is specified as follows. The **`messageType`** is set to '0' and the message is protected by an HMAC.

```
MessageBody() {
   RiId()                          // Defined in Appendix E.2.5
   TokenDeliveryId()               // Defined in Appendix E.2.10.4
   ReportingInformation()          // Defined in Appendix E.2.10.5
   TokenQuanity()                  // Defined in Appendix E.2.10.6
}
```

The fields are defined as follows:

- **`TokenQuantity`** – *Remaining Amount of BCAST Tokens* field in Table 74.

The message format (**`MessageBody`**) of the BCASTTokenEnablementResponse is specified as follows. The **`messageType`** is set to '1' and the message is protected by an HMAC.

```
MessageBody() {
   Status()                        // Defined in Appendix E.2.2
}
```

6.11.3.3.3 Exception Handling

There may be unexpected exceptions during the BCAST Token Enablement messages processing. The DRM SHOULD attempt to recover from the exception. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to Operation Log when a new MAKE process is executed.

[**Recovery Procedure – Local BCAST Token Consumption Recovery]**

The DRM Agent SHOULD re-attempt to perform BCAST Token Enablement procedure.

6.11.4 Retrieval of BCAST Token Information from the SRM

The DRM Agent can obtain information about the BCAST Tokens stored on the SRM using the BCAST Token Information Retrieval procedure illustrated in Figure 22. This protocol is intended for use in conjunction with BCAST Token Move from the SRM to Device and Local BCAST Token Consumption transactions.

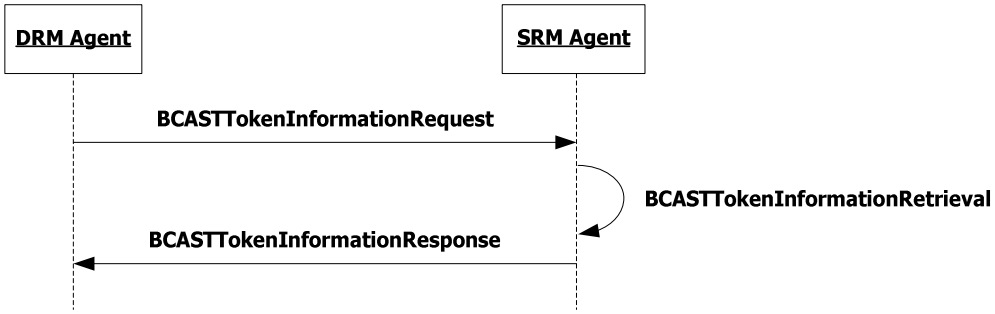

Figure 22: Sequence Diagram –BCAST Token Information Retrieval from SRM

6.11.4.1 Description of Messages

The DRM Agent sends BCASTTokenInformationRequest to the SRM Agent to request BCAST Token Information from SRM. The fields of the request are defined in Table 77.

Table 77: Fields of BCASTTokenInformationRequest

Fields	Protection Requirement	Description
RI ID	Integrity	Refer to section 5.1.10. If this element is present, only information on BCAST Tokens issued from this RI is returned.
Service/Program IDs	Integrity	Refer to section 5.1.10. If this element is present, only information on BCAST Tokens issued for this Service/Program is returned.

Upon receiving the BCASTTokenInformationRequest, the SRM Agent SHALL perform the following actions:

1. Verify the integrity of the fields in BCASTTokenInformationRequest.
2. Retrieve requested information from the SRM based on the search criteria defined in BCASTTokenInformationRequest.

The SRM Agent sends the BCASTTokenInformationResponse to carry the result of the procedure. The fields of the response are defined in Table 78.

Table 78: Fields of BCASTTokenInformationResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the BCASTTokenInformationRequest. The *Status* values are specified in Table 79.
RI ID	Integrity	Refer to section 5.1.10
Token Delivery ID	Integrity	Refer to section 5.1.10
Service ID/Program IDs	Integrity	Refer to section 5.1.10
Movable	Integrity	Refer to section 5.1.10
Domain IDs	Integrity	Refer to section 5.1.10
Latest Token Consumption Time	Integrity	Refer to section 5.1.10
Token Quantity	Integrity	Refer to section 5.1.10

Table 79: Values of *Status* field of the BCASTTokenInformationResponse

Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Token Not Found	The Token Delivery ID is not found on SRM.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response.

6.11.4.2 Format of Messages

The message format (**`MessageBody`**) of the BCASTTokenInformationRequest is specified as follows. The **`messageType`** is set to '0' and the message is protected by an HMAC.

```
MessageBody() {
    RiId()                          // Defined in Appendix E.2.5
    ServiceProgramIds()             // Defined in Appendix E.2.10.1
}
```

The message format (**`MessageBody`**) of the BCASTTokenInformationResponse is specified as follows. The **`messageType`** is set to '1' and the message is protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        nbrOfBCASTTokens    8       uimsbf
        for (i = 0; i < nbrOfBCASTTokens; i++)
        {
            RiId()                      // Defined in Appendix E.2.5
            TokenDeliveryId()           // Defined in Appendix E.2.10.4
            ServiceProgramIds()         // Defined in Appendix E.2.10.1
            Movable()                   // Defined in Appendix E.2.10.2
            DomainIds()                 // Defined in Appendix E.2.10.3
            LatestTokenConsumptionTime  // Defined in Appendix E.2.10.4
            TokenQuantity()             // Defined in Appendix E.2.10.6
        }
}
```

The fields are defined as follows:

- **nbrOfBCAST Tokens** is the number of BCAST Tokens retrieved from the SRM based on the search criteria specified in BCASTTokenInformationRequest.

6.11.5 BCAST Token Removal from the SRM

The DRM Agent can request the SRM Agent to delete certain BCAST Token using the BCAST Token Removal from the SRM procedure illustrated in Figure 23. BCAST Token Removal can be initiated upon the request from user. The DRM Agent uses this procedure within BCAST Token Move from the Device to SRM transaction.

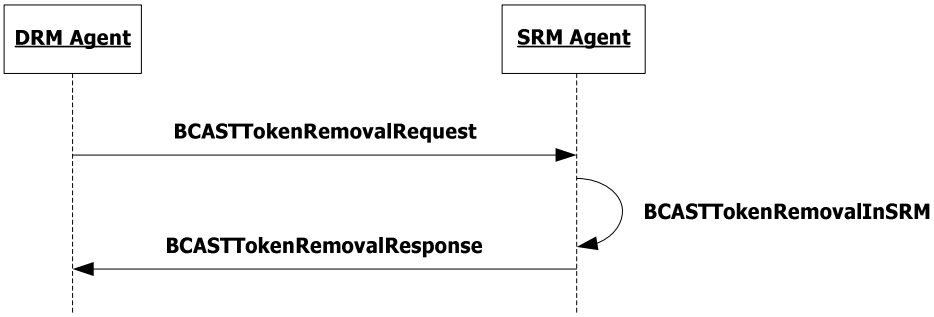

Figure 23: Sequence Diagram –BCAST Token Removal from the SRM

6.11.5.1 Description of Messages

The DRM Agent sends BCASTTokenRemovalRequest to the SRM Agent to request BCAST Token removal from SRM. The fields of the request are defined in Table 80.

Table 80: Fields of BCASTTokenRemovalRequest

Fields	Protection Requirement	Description
RI ID	Integrity	Refer to section 5.1.10
Token Delivery ID	Integrity	Refer to section 5.1.10

Upon receiving the BCASTTokenRemovalRequest, the SRM Agent SHALL perform the following actions:

1. Verify the integrity of the fields in BCASTTokenRemovalRequest.
2. Locate BCAST Token on the SRM using RI ID and BCAST Token Delivery ID from the BCASTTokenRemovalRequest.
3. Perform BCAST Token Removal by the SRM.

The SRM Agent sends the BCASTTokenRemovalResponse to carry the result of the procedure. The fields of the response are defined in Table 81.

Table 81: Fields of BCASTTokenRemovalResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the BCASTTokenRemovalRequest. The *Status* values are specified in Table 82.

Table 82:Values of *Status* field of the BCASTTokenRemovalResponse

Value	Description
Success	The request was successfully processed.

Value	Description
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Token Not Found	The BCAST Tokens are not found on SRM.
Remove Failed	The SRM Agent failed to Remove BCAST Token on SRM.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response.

6.11.5.2 Format of Messages

The message format (**`MessageBody`**) of the BCASTTokenRemovalRequest is specified as follows. The **`messageType`** is set to '0' and the message is protected by an HMAC.

```
MessageBody() {
    RiIds()                         // Defined in Appendix E.2.5
    TokenDeliveryID()               // Defined in Appendix E.2.10.4
}
```

The message format (**`MessageBody`**) of the BCASTTokenRemovalResponse is specified as follows. The **`messageType`** is set to '1' and the message is protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
}
```

6.11.5.3 BCAST Token Removal In SRM

6.11.5.3.1 Action Description

The SRM Agent removes the BCAST Tokens from the source SRM permanently.

6.11.5.3.2 Exception Handling

If the BCAST Tokens cannot be removed from the SRM, the SRM Agent SHALL generate relevant error code.

6.11.6 BCAST Token Upgrade

The DRM Agent can upgrade BCAST Tokens stored on the SRM using the BCAST Token Upgrade procedure illustrated in Figure 24. BCAST Token Upgrade SHALL be used for updating associated Reporting Information on the SRM, and MAY be used for accumulating BCAST Tokens on the SRM under the same RI ID and BCAST Token Delivery ID.

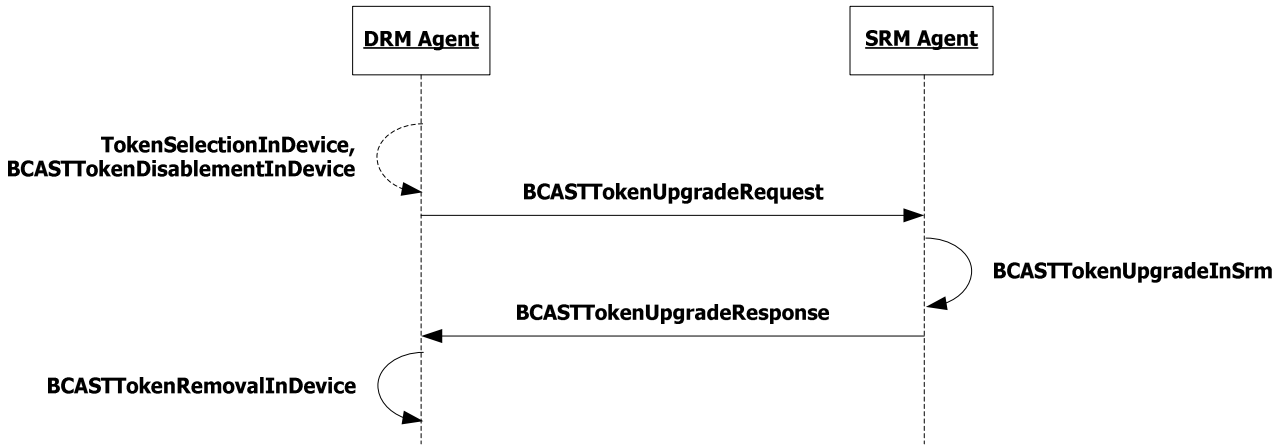

Figure 24: Sequence Diagram –BCAST Token Upgrade

6.11.6.1 Description of Messages

The DRM Agent sends BCASTTokenUpgradeRequest to the SRM Agent to request BCAST Token Upgrade in the SRM. The fields of the request are defined in Table 83.

Table 83: Fields of BCASTTokenUpgradeRequest

Fields	Protection Requirement	Description
RI ID	Integrity	Refer to section 5.1.10.
Token Delivery ID	Integrity	This Token Delivery ID is used to identify BCAST Token stored on the SRM. Token Delivery ID is defined in section 5.1.10.
New Token Delivery ID	Integrity	Specifies new Token Delivery ID to be set instead of the value stored in SRM. Token Delivery ID is defined in section 5.1.10.
Reporting Information	Integrity	Specifies new Reporting Information (e.g. Latest Token Consumption Time) to be set instead of the value stored in SRM. Reporting Information is defined in section 5.1.10.
Token Quantity	Integrity	Specifies new Token Quantity to be set instead of the value stored in SRM. Token Quantity is defined in section 5.1.10. Note: If this field is present in the request, the DRM Agent MUST disable the BCAST Tokens (see BCASTTokenDisablementInDevice) in the Device prior to sending BCASTTokenUpgradeRequest. If BCAST Tokens cannot be disabled, the DRM Agent MUST abandon BCAST Token Upgrade.

Upon receiving the BCASTTokenInformationRequest, the SRM Agent SHALL perform the following actions:

1. Verify the integrity of the fields in BCASTTokenUpgradeRequest.
2. Upgrade BCAST Tokens stored on the SRM based on new values given in BCASTTokenUpgradeRequest.

The SRM Agent sends the BCASTTokenUpgradeResponse to carry the result of the procedure. The fields of the response are defined in Table 84.

Table 84: Fields of BCASTTokenUpgradeResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the BCASTTokenUpgradeRequest. The *Status* values are specified in Table 85.

Table 85: Values of *Status* field of the BCASTTokenUpgradeResponse

Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Token Not Found	The Token Delivery ID is not found on SRM.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response. If no errors or exceptions occur (*Status* = *Success*), the DRM Agent SHALL continue with section 6.11.6.3 (BCAST Token Removal in the Device).

6.11.6.2 Format of Messages

The message format (**`MessageBody`**) of the BCASTTokenUgradeRequest is specified as follows. The **`messageType`** is set to '0' and the message is protected by an HMAC.

```
MessageBody() {
    TokenQuantityPresent     1      uimsbf
    Rfu                     7       uimsbf
    RiId()                          // Defined in Appendix E.2.5
    TokenDeliveryId()               // Defined in Appendix E.2.10.4
    NewTokenDeliveryId()            // Defined in Appendix E.2.10.4
    ReportingInformation()          // Defined in Appendix E.2.10.5
    if ( TokenQuantityPresent )
    {
        TokenQuantity()             // Defined in Appendix E.2.10.6
    }
}

NewTokenDeliveryID()
{
    TokenDeliveryID()               // Defined in Appendix E.2.10.4
}
```

The fields are defined as follows:

- **`TokenQuantityPresent`** – this flag indicates presence of Token Quantity attribute in this data structure. If it is present, this flag is set to '1'.

The message format (**MessageBody**) of the BCASTTokenUpgradeResponse is specified as follows. The **messageType** is set to '1' and the message is protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
}
```

6.11.6.2.1 Exception Handling

There may be unexpected exceptions during the BCAST Token Upgrade messages processing. The DRM SHOULD attempt to recover from the exception. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to Operation Log when a new MAKE process is executed.

[**Recovery Procedure –BCAST Token Upgrade Recovery]**

The DRM Agent SHOULD attempt to recover BCAST Token Upgrade transaction. In that case, the DRM Agent MAY check if BCAST Token Upgrade was successfully performed in the SRM by using BCAST Token Information Retrieval procedure. If BCAST Token Upgrade was successful, the DRM Agent SHALL proceed to BCAST Token Removal in the Device defined in section 6.11.6.3. Otherwise, the DRM Agent MAY restart or cancel BCAST Token Upgrade procedure.

After recovering the BCAST Token Upgrade transaction, the DRM MUST remove operational log entry for this procedure.

6.11.6.3 BCAST Token Removal in the Device

6.11.6.3.1 Action Description

The DRM Agent removes the BCAST Tokens from the source Device permanently.

6.11.6.3.2 Exception Handling

There may be unexpected exceptions during BCAST Token Removal in Device. The DRM Agent SHOULD attempt to recover from the exception as described in paragraph 6.11.6.2.1.

6.11.7 Movement of Broadcast Rights from Device to SRM

Broadcast Rights are Moved from a Device to an SRM as illustrated in

Figure 25. This transaction is comprised of two request-response message pairs: Installation Setup message pair (as defined in section 6.11.7.1) and Broadcast Rights Installation message pair.

Before sending InstallationSetupRequest message the DRM MUST do the following,

1. Per each selected asset (contained in BCRO base), check if Move transaction is allowed by checking presence of *move* action type in OMADRMAction() object (this object is defined in [DRMXBSv1.1]). If Move is not allowed for the current asset(s), then do not include this asset in BCRO Assets data defined in section 5.1.11.2. If Move is not allowed for any asset in the BCRO then do not perform Move transaction.
2. Per each *move* action check if the *system* constraint is present in the OMADRMAction() object. If it is present then check value of the *system_id* attribute. If *system_id* subfield indicates the SRM protocol then add this asset(s) in BCRO Asset data.

Before sending the BroadcastRightsInstallationRequest message, the DRM Agent MUST do the following:

1. Per each *move* action check if the *count* constraint is present in the OMADRMAction() object. If it is present check in the state information the current value of the Move Count that is associated with the action. If the value is "0" then do not perform the Move transaction for the current asset(s). Otherwise (Move Count > 0), decrement the Move Count value by 1.

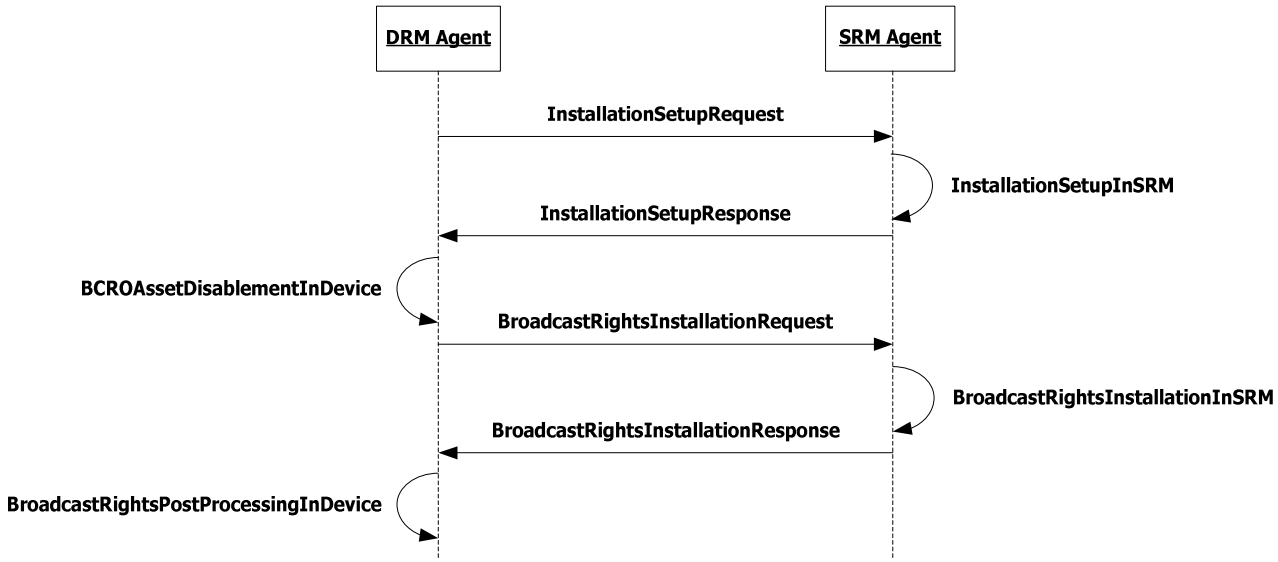

Figure 25: Sequence Diagram – Movement of Broadcast Rights from Device to SRM

6.11.7.1 Installation Setup

The DRM agent SHALL use Installation Setup procedure defined in section 6.5.1 with the following adaptation:

- *Size of Rights* field in the InstallationSetupRequest message MUST be the size of Broadcast Rights that will be installed in the SRM
    - *Size of Rights* = Length of **`BcroBase +`** Length of **`BcroAssets +`** Length of **`BcroStateInfo`** `(if present)` **`+`** `Length of` **`BcroSignature`** `(if present).`

6.11.7.2 BCRO Asset Disablement in Device

The DRM agent SHALL use Rights Disablement procedure defined in section 6.5.2 with the following adaptation:

- The DRM Agent SHALL only disable the use of BCRO Assets that will be Moved within the current transaction. These assets can only be enabled when Moved back from SRM to Device or during transaction recovery.

6.11.7.3 Broadcast Rights Installation

6.11.7.3.1 Description of Messages

The DRM Agent sends the BroadcastRightsInstallationRequest to install the Broadcast Rights in the SRM. The fields of the request are defined in Table 86.

Table 86: Fields of BroadcastRightsInstallationRequest

Fields	Protection Requirement	Description

Fields	Protection Requirement	Description
Handle	Integrity & Confidentiality	Same as the Handle transmitted by the InstallationSetupRequest in 6.11.7.1. Refer to section 5.1.3.
BCRO Base	Integrity	Refer to section 5.1.11.1.
BCRO Assets	Integrity & confidentiality	Refer to section 5.1.11.2.
BCRO State Info	Integrity	Refer to section 5.1.11.3.
BCRO Signature	Integrity	Refer to section 5.1.11.4.

Upon reception of the BroadcastRightsInstallationRequest, the SRM Agent installs the BroadcastRights in the SRM. For the installation, the SRM Agent MUST perform the following procedure:

1. Verify the integrity of the request fields
2. Decrypt the *Handle* and *BCRO Assets* with the Session Key
3. Compare the *Handle* with the *Handle* in the InstallationSetupRequest
4. Install the *BCRO Base, BCRO Assets , BCRO State Info* and *BCRO Signature* at a space associated with the *Handle*.

The SRM Agent sends the BroadcastRightsInstallationResponse to carry the result of the procedure. The fields of the response are defined in Table 87.

Table 87: Fields of BroadcastRightsInstallationResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the BroadcastRightsInstallationRequest message. The *Status* values are specified in Table 88.

Table 88: Status of Broadcast Rights Installation Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Handle Not Found	The Handle in the request does not exist in the SRM.
Handles In-consistent	The Handle in this request is different from the *Handle* in the InstallationSetupRequest.
Not Enough Space	The size of *BCRO Base* exceeds *Size of Rights* in InstallationSetupRequest.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response.

If no errors or exceptions (*Status* = *Success*), the DRM Agent continues with section 6.11.9.

6.11.7.3.2 Format of Messages

The message format (**`MessageBody`**) of the BroadcastRightsInstallationRequest is specified as follows. The **`messageType`** is set to '0' and the message is protected by an HMAC.

```
MessageBody() {
    EncryptedHandle()               // Defined in Appendix E.2.8
    BcroBase()                      // Defined in Appendix E.2.11.1
    BcroAssets()                    // Defined in Appendix E.2.11.2
    BcroStateInfo()                 // Defined in Appendix E.2.11.3
    BcroSignature()                 // Defined in Appendix E.2.11.4
}
```

The fields are defined as follows:

- **`EncryptedHandle`** – *Handle* encrypted with the current Session Key
- **`BCROBase`** – *BCRO Base* field in Table 86
- **`BCROAsset`** – *BCRO Assets* field in Table 86
- **`BCROStateInfo`** – *BCRO State Info* field in Table 86
- **`BCROSignature`** – *BCRO Signature* field in Table 86

The message format (**`MessageBody`**) of the BroadcastRightsInstallationResponse is specified as follows. The **`messageType`** is set to '1' and the message is protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
}
```

The field is defined as follows:

- **`Status`** - *Status* field in Table 87

6.11.7.3.3 Exception Handling

The DRM Agent SHALL follow the procedures defined in section 6.5.3.3.

6.11.7.4 Post-processing of Broadcast Rights in Device

6.11.7.4.1 Action Description

The DRM Agent MAY remove the Broadcast Rights from the source Device permanently. However, due to a broadcast nature of these rights, they may be delivered to a Device multiple times. Duplication of Broadcast Rights is avoided by application of the Move Cache, as specified in Chapter 9.

In addition, if some of assets in the BCRO were not Moved to the SRM (within BCRO Assets), the DRM Agent SHALL not remove the Broadcast Rights from the Device.

The Assets that have been Moved MUST be removed.

When this action is completed, the Move is terminated and the entry for the Move transaction is removed from the Operation Log.

6.11.7.4.2 Exception Handling

If the removal process is performed and cannot be complete, the DRM Agent SHALL follow the procedure defined in section 6.5.4.2.

6.11.8 Movement of Broadcast Rights from SRM to Device

Broadcast Rights are Moved from an SRM to a Device as illustrated in Figure 26. This transaction is comprised of two request/response message pairs: Broadcast Rights Retrieval message pair and Rights Removal message pair. The Broadcast Rights Retrieval message pair MAY be followed by the Rights Removal message pair. If no BCRO Asset remains enabled after the Broadcast Rights Retrieval exchange, Rights Removal SHALL be performed.

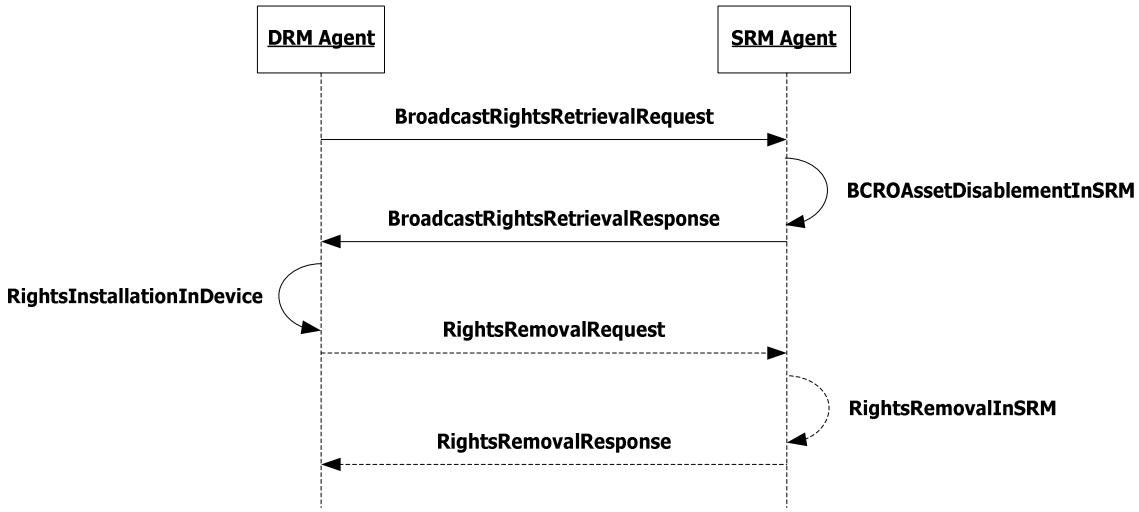

Figure 26: Sequence Diagram – Movement of Broadcast Rights from SRM to Device

6.11.8.1 Broadcast Rights Retrieval

6.11.8.1.1 Description of Messages

The DRM Agent sends the BroadcastRightsRetrievalRequest to initiate the Move of the Broadcast Rights from the SRM. The fields of the request are defined in Table 89.

Table 89: Fields of BroadcastRightsRetrievalRequest

Fields	Protection Requirement	Description
Handle	Integrity	This identifies (Broadcast) Rights that will be Moved from the SRM to the Device. Refer to the section 5.1.3.
New Handle	Integrity & Confidentiality	*New Handle* is a 10 byte random value generated by the DRM Agent for this Move transaction.
Asset Index List	Integrity	*Asset Index List* is a list of indexes of assets selected for the current Move transaction. This field is optional, and if it is not included complete list of BCRO Assets SHALL be delivered in BroadcastRightsRetrievalResponse. Asset index is defined in section 5.1.11.2.

Upon receiving the BroadcastRightsRetrievalRequest, the SRM Agent MUST perform the following procedure:

1. Verify the integrity of the request fields
2. Find Broadcast Rights corresponding to the *Handle*
3. If found, then decrypt the *New Handle* with the Session Key
4. Check if the SRM already has the same Handle with *New Handle*. If yes, the SRM Agent sets *Status* to *Duplicate Handle* and sends BroadcastRightsRetrievalResponse to the DRM Agent. If no, overwrite the *Handle* in the SRM with the *New Handle*, and proceed to next step.

5. Disable BCRO Assets identified in the *Asset Index List* field of the request. If *Asset Index List* field is not present, then disable all BCRO Assets associated to the current Handle.

The SRM Agent sends the BroadcastRightsRetrievalResponse to carry the result of the procedure. The fields of the response are defined in Table 90.

Table 90: Fields of BroadcastRightsRetrievalResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the BroadcastRightsRetrievalRequest message. The *Status* values are specified in Table 91. If *Status* contains any error, only this field is present in the BroadcastRightsRetrievalResponse.
BCRO Base	Integrity	Refer to section 5.1.11.1.
BCRO Assets	Integrity & Confidentiality	Refer to section 5.1.11.2. Contains the list of BCRO Assets as requested in BroadcastRightsRetrievalRequest.
BCRO State Info	Integrity	Refer to section 5.1.11.3.
BCRO Signature	Integrity	Refer to section 5.1.11.4.

Table 91: Status of Broadcast Rights Retrieval Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Handle Not Found	The Handle in the request does not exist in the SRM.
Duplicate Handle	The SRM already has the *New Handle* and its corresponding Rights.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent MUST perform the following procedure:

1. Verify the integrity of fields in the response
2. Decrypt *BCRO Assets* contents with the Session Key

If no errors or exceptions (*Status* = *Success*), the DRM Agent continues with section 6.6.2.

6.11.8.1.2 Format of Messages

The message format (**`MessageBody`**) of the BroadcastRightsRetrievalRequest is specified as follows. The **`messageType`** is set to ’0’ and the message is protected by an HMAC.

```
MessageBody() {
    Handle()                        // Defined in Appendix E.2.4
    EncryptedNewHandle()            // Defined in Appendix E.2.9
    AssetIndexListPresent    1      uimsbf
    rfu                      7      uimsbf
    if ( AssetIndexListPresent )
    {
        AssetIndexList()            // Defined in Appendix E.2.11.2.6
    }
}
```

The fields are defined as follows:

- **`Handle`** –*Handle* field in Table 89
- **`EncryptedNewHandle`** – *New Handle* field in Table 40 encrypted with the current Session Key (SK)
- **`AssetIndexListPresent`** – indicates presence of Asset Index List in this message
- **`AssetIndexList`** – *`Asset Index List`* `field in` Table 89

The message format (**`MessageBody`**) of the BroadcastRightsRetrievalResponse is specified as follows. The **`messageType`** is set to '1' and the message is protected by an HMAC.

```
MessageBody()
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
      BcroBase()                    // Defined in Appendix E.2.11.1
      BcroAssets()                  // Defined in Appendix E.2.11.2
      BCROStateInfo()               // Defined in Appendix E.2.11.3
      BCROSignature()               // Defined in Appendix E.2.11.4
    }
}
```

The fields are defined as follows:

- **`Status`** - *Status* field in Table 90

6.11.8.1.3 Exception Handling

The DRM Agent SHALL follow the procedure defined in section 6.6.1.3.

6.11.8.2 BCRO Asset Disablement in the SRM

6.11.8.2.1 Action Description

The SRM Agent disables assets identified in the BroadcastRightsRetrievalRequest. The disabled BCRO Asset cannot be used for the other purposes except the current Move transaction.

6.11.8.2.2 Exception Handling

When the exception occurs, the DRM Agent SHOULD immediately attempt to recover from it. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to the Operation Log when a new MAKE process is executed.

6.11.8.3 BroadcastRights Installation in Device

6.11.8.3.1 Action Description

The DRM Agent performs the following procedure:

1. The RI signature over the Broadcast Rights SHOULD be verified if present (note that signature verification MAY not be required under certain trust models as described in section 5.1.2). If the verification fails, the DRM Agent MUST re-enable the Broadcast Rights on the SRM as described in section 6.11.10.4 and terminate the Move transaction.

2. Per each asset delivered in BroadcastRightsRetrievalResponse, check if the *move* action *count* constraint is present in the OMADRMAction() object. If it is present, do the following:

   A. If in the state information, the Move Count associated with the action is non-zero, it is decremented by one.

   B. Otherwise (the Move Count is zero), the DRM Agent MUST re-enable Broadcast Rights on the SRM as described in section 6.11.10.4 and terminate the Move transaction.

3. Per each move action check if the *system* constraint is present in the OMADRMAction() object. If it is present, check the value of the *system_id* attribute. If *system_id* subfield indicates a protocol other than the SRM protocol, the constraint is not met, and therefore the DRM agent cannot use this action to perform BCRO Move. If no action is available that either has no *system* constraint or has a *system* constraint with a *system_id* indicating the SRM protocol, the DRM Agent MUST re-enable the Broadcast Rights on the SRM as described in section 6.11.10.4 and terminate the Move transaction.

4. Install the Broadcast Rights wi*th th*e following conditions:

   A. Broadcast Rights received via the Move protocol SHALL NOT be rejected based on the content of any DRM V2.0 replay cache. The replay cache is specified in section 9.4 of [OMADRMv2.0].

   B. The Broadcast Rights SHALL NOT be installed if a BCRO with the same enabled asset(s) in the OMADRMAsset() object (see [DRMXBSv1.1]) is already installed on the device. If so, then the DRM Agent MUST re-enable the Broadcast Rights on the SRM as described in section 6.11.10.4 and terminate the Move transaction.

   C. If a BCRO with the same asset(s) is already installed but the asset is disabled in the Device, the DRM Agent MUST enable the Broadcast Rights on the Device and update asset state information based on the data received from the SRM.

After the Broadcast Rights installation, the DRM Agent MAY continue with section 6.11.8.4.

6.11.8.3.2 Exception Handling

There may be unexpected exceptions when installing Broadcast Rights. The exception causes the installation to not complete.

When an exception occurs, the DRM Agent SHOULD immediately attempt to recover from it. If the DRM Agent fails to detect the exception, it MUST recover from the exception by referring to the Operation Log when a new MAKE process is executed.

6.11.8.4 Broadcast Rights Removal

The DRM Agent SHALL use Rights Removal procedure defined in section 6.6.3.1.

6.11.9 Local BCAST Rights Consumption

This section defines a mechanism which allows a Device to consume BCAST Rights that require presence of a SRM that has the BCAST Rights (as indicated by the <srmPing> constraint, see section I.2.1). This mechanism makes use of the Local Rights Consumption mechanism (section 6.7) and SRM Ping protocol (section 6.11.9.1) as illustrated in Figure 27.

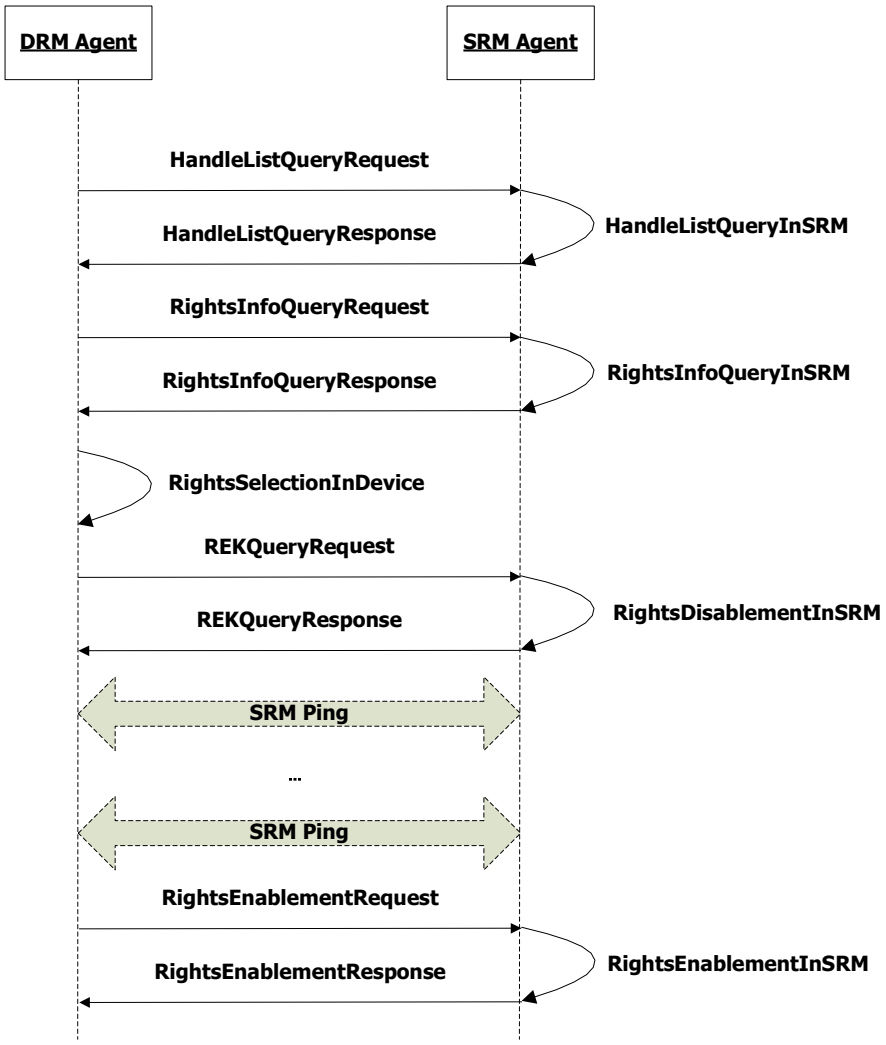

Figure 27: Sequence Diagram – Local BCAST Rights Consumption

The consumption of Rights requiring presence of SRM is specified as follows:

1. Retrieval of Rights from the SRM
    a) If the DRM Agent knows a service_CID or program_CID of the BCAST content (see [DRMXBSv1.1]), corresponding *Handle* SHALL obtained from the SRM using the Handle List Query transaction. The Handle List Query Request message SHALL include Asset ID that is the BCAST service_CID or program_CID as defined in section 5.1.5.

b) If unknown, corresponding *Handle* SHALL obtained from the SRM using Handle List Query mechanism (see section 6.12.1)

c) *Rights Information* SHALL be obtained from SRM using Rights Information Query mechanism (see section 6.12.2). At this moment the DRM Agent SHALL store the SRM ID of the SRM provided this Rights Information. The DRM Agent MAY check whether the Rights Information is associated to the BCAST service or not.

d) REK Query mechanism (section 6.7.2) SHALL be used to retrieve REK from the SRM. Upon reception of the REK Query Response, the REK is used for decryption of the SEK/PEK which is contained in the BCAST Rights (see [DRMXBSv1.1]).

2. Consumption of Rights and SRM Ping protocol

a) During the consumption of BCAST Rights, the SRM Ping protocol defined in section 6.11.9.1 MUST be executed if <srmPing> constraint is present in the associated permission. SRM Ping protocol is used to check the presence of the SRM that provided a copy of the BCAST Rights to the DRM Agent (as indicated by the stored SRM ID) and still contains the BCAST Rights. Execution of the SRM Ping protocol SHALL be synchronised with the BCAST STKM processing or executed repeatedly as specified in the <srmPing> constraint. (See the section I.2.1 for details).

b) If SRM Ping protocol finishes with failure or cannot be completed, the DRM Agent MUST cease (or not initiate) the consumption of BCAST Rights with the <srmPing> constraint. In addition, the DRM Agent SHALL delete the local copy of the BCAST Rights and its associated REK, and proceed to step 3a.

c) If SRM Ping protocol is not implemented by the DRM Agent, the DRM Agent MUST not initiate the consumption of BCAST Rights with the <srmPing> constraint.

3. Consumption of Rights completion.

a) When the consumption of BCAST Rights is completed, the DRM Agent SHALL stop execution of SRM Ping Protocol, SHALL enable the BCAST Rights on the SRM using Rights Enablement mechanism (section 6.12.5), and SHALL delete the local copy of the BCAST Rights and its associated REK.

6.11.9.1 SRM Ping

SRM Ping protocol is an optional feature for checking the presence of SRM that contains necessary BCAST Rights.

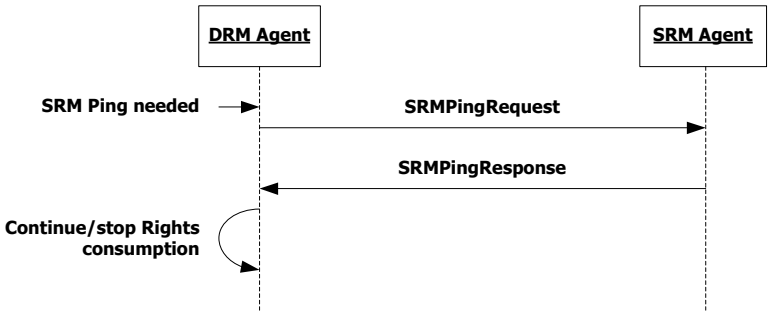

Figure 28: Sequence Diagram - SRM Ping

The SRM Ping protocol is used for the verification of the presence of a SRM that has the BCAST Rights. When a permission contains an <srmPing> constraint (see section I.2.1), the permission can only be exercised as long as the SRM Ping protocol can be executed. SRM Ping protocol implements two modes of operation, where the mode depends on <srmPing> constraint subelements. In the first mode, the protocol is executed synchronously with the BCAST STKM processing (see [DRMXBSv1.1]) as indicated by <synchronised> subelement of <srmPing> constraint. In the second mode, the protocol is executed repeatedly (regardless of the BCAST STKM processing), where the time between two subsequent executions is specified by the <checkInterval> subelement of the <srmPing> constraint.

The DRM Agent MAY support the SRM Ping protocol. However, the DRM Agent which supports BCAST STKM processing procedure SHALL support the SRM Ping protocol. If the SRM Ping protocol is not supported, permissions with the <srmPing> constraint MUST NOT be granted. If the <srmPing> constraint contains the <checkInterval> subelement, the DRM Agent MUST support the DRM Time. In the remainder of this section and its subsections, it is assumed that both the DRM Agent and the SRM Agent support the SRM Ping protocol.

The SRM Ping protocol can only be executed between the DRM Agent and the SRM which stores the RO containing <srmPing> constraint. As a requirement for the execution of the SRM Ping protocol, a SAC MUST be established between the DRM Agent and the SRM Agent.

Figure 28 describes the flow of the SRM Ping protocol. The protocol is initiated by the DRM Agent sending a SRMPingRequest message to the SRM Agent. After the SRM Agent verifies the presence of the BCAST Rights, the SRM Agent returns a SRMPingResponse message to the DRM Agent. If the SRMPingResponse is valid, the DRM Agent can continue/initiate consumption of the permission, otherwise it stops the consumption.

6.11.9.1.1 Description of Messages

By default, the message body of the SRMPingRequest message does not contain any fields. However, the SRMPingRequest MAY contain the *Handle* and the *RAND* fields in the case that there is a demand from Trust Authority.

The Trust Authority MAY mandate the inclusion of the Handle and RAND parameters. Usage of the parameters guarantees the verification that the SRM Agent has knowledge of the REK during executing the SRM Ping Protocol.

In case that SRMPingRequest contains the Handle and RAND parameters, before sending the SRMPingRequest, the DRM Agent SHALL store the RAND that will be used for checking the *Hash Data* when the DRM Agent receives the SRMPingResponse. The fields of the request for this case are described in Table 92.

Table 92: Fields of SRMPingRequest

Fields	Protection Requirement	Description
Handle	Integrity	*Handle* identifies Rights that will be checked in the SRM.
RAND	Integrity	*RAND* is a random number generated by the SRM Agent.

The request is protected by an HMAC. Upon reception of an SRMPingRequest, the SRM Agent MUST perform the following procedure:

1. Verify if the SRMPingRequest message was well formed and has integrity protection using the HMAC. If verification fails, the SRM Agent MUST return an SRMPingResponse that contains the *Status* field set to Field Integrity Verification Failed.
2. If *Handle* and *Rand* fields are not included in the request, the SRM Agent MUST return the SRMPingResponse with the *Status* field set to *Success*.
3. If the SRMPingRequest contains *Handle* and *RAND* fields, SRM Agent MUST find the Rights corresponding to the *Handle*. If the SRM Agent does not find the valid Rights, the SRM Agent MUST return a SRMPingResponse that contains the *Status* field set to *Handle Not Found*. Else the SRM Agent returns a SRMPingResponse that contains the *Hash Data* field that contains hash value of the concatenation of *RAND* and *REK*, the *Status* field set to *Success*.

The fields of the SRMPingResponse message are described in Table 90. The possible values of the *Status* field are listed in Table 91.

Table 93: Fields of SRMPingResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the SRMPingRequest. The *Status* values are specified in Table 94. If *Status* contains any error, only this field is present in the SRMPingResponse.
Hash Data	Integrity	H(RAND | REK). The *RAND* is received from the DRM

Fields	Protection Requirement	Description
		Agent in the SRMPingRequest. The *REK* is associated with the Rights for BCAST service. This field is only present when the request contains *Handle* and *RAND* fields, and *Status* value of the response is set to *Success*.

Table 94: Status of SRMPingResponse message

Status Value	Description
Success	The request was successfully processed
Field Integrity Verification Failed	The HMAC value of the request did not match the HMAC value generated by the SRM Agent.
Handle Not Found	The Handle in the request does not exist in the SRM.
Parameter Failed	A field in the request has an invalid length or structure.
Unknown Error	Other errors

Upon receipt of the SRMPingResponse with the *Status* field set to *Success*, the DRM Agent MUST perform the following procedure:

1. Verify the integrity of the response.
2. If the verification is successful and the SRMPingRequest contained the Handle and the RAND parameter, the DRM Agent checks the *Hash Data* in the response message. In order to check the Hash Data, the DRM Agent generates the Hash of *RAND* and *REK*, where the *RAND* is same as contained in SRMPingRequest, and the *REK* is associated to the local copy of BCAST Rights.

If the HMAC verification fails, Hash Data verification fails or the *Status* field in the SRMPingResponse was not set to *Success*, or the Device did not receive any valid SRMPingResponse within a certain time period, the SRM Ping protocol has failed and the DRM Agent MUST cease (or not initiate) the consumption of the permission associated with the <srmPing> constraint.

Otherwise, the DRM Agent can continue/initiate the consumption of the permission, given that other constraints in the RO are also satisfied.

6.11.9.1.2 Format of the messages

The message format (**`MessageBody`**) of the SRMPingRequest is specified as follows. The **`messageType`** is set to'0' and the message is protected by an HMAC.

```
RAND(){
    RandomNumber()                  // Defined in Appendix E.1
}
```

```
MessageBody() {
    HandleRandPresent        1      uimsbf
    rfu                      7      uimsbf
    if ( HandleRandPresent == 1 )
    {
        Handle()                    // Defined in Appendix E.2.4
        RAND()
    }
}
```

The fields are defined as follows:

- **`Handle`** – *Handle* field in Table 89
- **`RAND`** – *RAND* field in Table 89

The message format (**`MessageBody`**) of the SRMPingResponse is specified as follows. The **`messageType`** is set to '1' and the message is protected by an HMAC.

```
HashData() {
    // SHA-1 hash of RAND and REK
    Hash()                          // Defined in Appendix E.1
}
```

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        HashData()
    }
}
```

The fields are defined as follows:

- **`Status`** - *Status* field in Table 93.
- **`HashData`** - *Hash Data* field in Table 93

6.11.10 Utility protocols for Broadcast Rights management

6.11.10.1 BCAST Extensions to Handle List Query

The DRM Agent SHALL use Handle List Query procedure specified in section 6.12.1 to read a Handle List associated to the BCRO(s). When using this procedure, the DRM Agent MUST include the following extension in the HandleListQueryRequest:

```
Extension() {
    nbrOfBci                 8      uimsbf
    for ( i=0 ; i<nbrOfBci ; i++ ) {
        Bci()                       // Defined in Appendix E.2.11.2.2
    }
    HandleListLength        16      uimsbf
}
```

The **extensionType** in the **ExtensionsContainer()** (see section 5.6.5) for this extension SHALL be set to '0'. The fields of the extension are defined as follows:

- **nbrOfBci** – number of BCIs
- **Bci** – *BCI* defined in section 5.1.11.2
- **HandleListLength** – *Handle List Length*

If this extension is included, the **nbrOfAssetId** field in the HandleListQueryRequest SHALL be set to '0'.

6.11.10.2 Retrieval of BCRO Information from the SRM

The DRM Agent can obtain information about the certain BCRO stored on the SRM using the BCRO Information Retrieval procedure illustrated in Figure 29. BCRO information includes BCRO Base and State Info pair. This protocol is intended for use in conjunction with BCRO Move from the SRM to Device transaction.

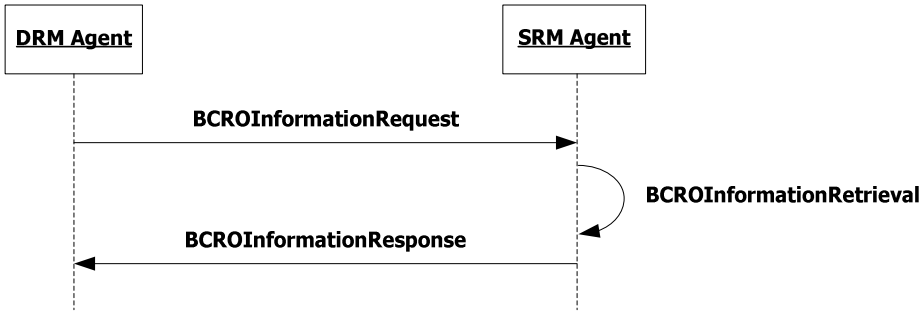

Figure 29: Sequence Diagram – BCRO Information Retrieval from SRM

6.11.10.2.1 Description of Messages

The DRM Agent sends BCROInformationRequest to the SRM Agent to request BCRO information from SRM. The fields of the request are defined in Table 95.

Table 95: Fields of BCROInformationRequest

Fields	Protection Requirement	Description
Handle	Integrity	This identifies the (Broadcast) Rights in the SRM. Refer to section 5.1.3.

Upon receiving the BCROInformationRequest, the SRM Agent MUST perform the following actions:

1. Verify the integrity of the request fields.
2. Find the BCRO corresponding to the Handle.

The SRM Agent sends the BCROInformationResponse to carry the result of the procedure. The fields of the response are defined in Table 96.

Table 96: Fields of BCROInformationResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the BCROInformationRequest. The *Status* values are specified in Table 97.
BCRO Base	Integrity	Refer to section 5.1.11.1
BCRO State Info	Integrity	Refer to section 5.1.11.3
Available Asset Index List	Integrity	This identifies the BCRO Assets that are enabled on the SRM. BCRO Assets are defined in section 5.1.11.2.

Table 97: Values of *Status* field of the BCROInformationResponse

Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Handle Not Found	The Handle in the request does not exist in the SRM.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response.

6.11.10.2.2 Format of Messages

The message format (**`MessageBody`**) of the BCROInformationRequest is specified as follows. The **`messageType`** is set to '0' and the message is protected by an HMAC.

```
MessageBody() {
    Handle()                        // Defined in Appendix E.2.4E.2.4
}
```

The message format (**`MessageBody`**) of the BCROInformationResponse is specified as follows. The **`messageType`** is set to '1' and the message is protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        BcroBase()                  // Defined in Appendix E.2.11.1
        BcroStateInfo()             // Defined in Appendix E.2.11.3
        AssetIndexList()            // Defined in Appendix E.2.11.2.7
    }
}
```

The fields are defined as follows:

- AssetIndexList **– Available Asset Index List field in Table 96**

6.11.10.2.3 Exception Handling

There may be an unexpected exception during the BCRO Information Retrieval Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response, finds an error by referring to the *Status* (except *Handle Not Found*), or fails to verify the integrity of fields, then the DRM Agent regards it as an exception and terminates communication with the SRM Agent. The User may be informed of the exception.

6.11.10.3 Retrieval of BCRO Information List from the SRM

The DRM Agent can obtain information about the BCROs stored on the SRM using the BCRO Information List Retrieval procedure illustrated in Figure 30. BCRO Information List includes a set of {BCRO Base, State Info} pairs.

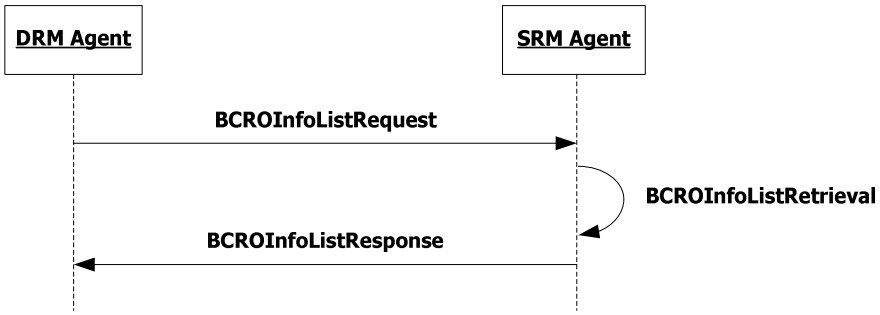

Figure 30: Sequence Diagram –BCRO Information List Retrieval from SRM

6.11.10.3.1 Description of Messages

The DRM Agent sends BCROInfoListRequest to the SRM Agent to request BCRO Information List from SRM. The fields of the request are defined in Table 98.

Table 98: Fields of BCROInfoListRequest

Fields	Protection Requirement	Description
BCI List	Integrity	List of BCIs which identifies relevant broadcast content. The DRM Agent MAY selectively request BCRO information by including broadcast content identifiers into BCROInfoListRequest. If this field is zero, information on all BCROs stored in the SRM is returned.
BCRO Information List Length	Integrity	Maximum list length (in bytes) that the DRM Agent can process. If this value is non-zero, the SRM Agent MUST send BCRO Information List shorter than or equal to this value. If BCRO information list is longer than this value, the SRM Agent SHALL divide the BCRO Information List into several chunks.

Upon receiving the BCROInfoListRequest, the SRM Agent performs the following actions:

- Verify the integrity of the request fields.
- If the number of BCIs in the *BCI List* is zero, the SRM Agent generates a BCRO Information List of all BCROs in the SRM.
- Otherwise, the SRM Agent generates a BCRO Information List of BCROs in the SRM that are associated with the requested DRM Contents identified by the BCIs in the request.

The SRM Agent sends the BCROInfoListResponse to carry the result of the action. The fields of the response are defined in Table 99. If the number of BCIs in the BCROInfoListRequest is more than the *Max Number Of AssetIDs* specified in the SrmHelloResponse, the SRM Agent MUST return the error - *AssetID List Too Long*.

Table 99: Fields of BCROInfoListResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the BCROInformationRequest. The *Status* values are specified in Table 100.
BCRO Information List	Integrity	This field contains a list of {BCRO Base, BCRO State Info} pairs. Note that this list can be divided into several chunks and delivered in multiple responses.
Continuation Flag	Integrity	It is assumed that a Rights Information List is divided into several chunks. '0': The BCRO Information List in this response is the last chunk of the whole BCRO Information List, or the BCRO Information List is not divided into chunks. '1': The BCRO Information List has been divided into several chunks. The *BCRO Information List* in this response is a chunk of the whole BCRO Information List, and there are subsequent chunks.

Table 100:Values of *Status* field of the BCROInfoListResponse

Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
AssetID List Too Long	The number of BCIs in the request exceeds the *Maximum Number Of AssetIDs*.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

If the Continuation Flag contains the value '1' and *Status* is *Success*, the DRM Agent SHOULD send the BCROInfoListRequest again, with the same field values, in order to read the next chunk. The DRM Agent repeats the BCROInfoListRequest until the response contains the value '0' in the *Continuation Flag* field. If the *Continuation Flag* contains the value '0' and no errors or exceptions (*Status* = *Success*), the BCROInfoListRequest message processing is completed.

If the DRM Agent sends a different message or sends the BCROInfoListRequest with different field values than the previous values, then the SRM Agent resets the operation (i.e. the SRM Agent returns the BCRO Information List from the first chunk again).

If a BCRO Information List is divided into several chunks, where a chunk will contain a portion of the complete list, the DRM Agent MUST concatenate all chunks in sequence from the SRM Agent in order to receive the complete BCRO Information List.

6.11.10.3.2 Format of Messages

The message format (**`MessageBody`**) of the BCROInfoListRequest is specified as follows. The **`messageType`** is set to '0' and the message is protected by an HMAC.

```
MessageBody() {
    nbrOfBci                        8   uimsbf
    for ( i = 0 ; i < nbrOfBci ; i++ ) {
        Bci()                       // Defined in Appendix E.2.11.2.2
    }
    BcroInfoListLength              16  uimsbf
}
```

The fields are defined as follows:

- **nbrOfBci** – Number of **Bci** in *BCI List* field
- **Bci** – *BCI* field in Table 98
- **BcroInfoListLength** – *BCRO Information List Length* field in Table 98

The message format (**MessageBody**) of the BCROInfoListResponse is specified as follows. The **messageType** is set to '0' and the message is protected by an HMAC.

```
MessageBody() {
    Status()                            // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        BcroInformationList()           // Defined in Appendix E.2.11.3
        continuationFlag            1   bslbf
        rfu                         7   bslbf
    }
}
```

The fields are defined as follows:

- **Status** - *Status* field in Table 99
- **BcroInformationList** – *Rights Information List* field in Table 99
- **continuationFlag** – *Continuation Flag* field in Table 99

6.11.10.3.3 Exception Handling

There may be an unexpected exception during the BCRO Information List Retrieval Message processing as specified in section 6.11.10.3. If the DRM Agent fails to receive the response or finds an error by referring to the *Status*, then the DRM Agent regards it as an exception and terminates the BCRO Information List Retrieval Message processing. The User MAY be informed of the exception.

6.11.10.4 BCRO Asset Enablement on the SRM

The DRM Agent can enable BCRO Asset(s) that are disabled on the SRM using Rights Enablement procedure defined in section 6.12.5. The following adaptations SHALL be made when using this procedure:

- *State Information* field in Table 113 (Fields of RightsEnablementRequest) SHALL be BCRO State Info defined in section 5.1.11.3.
- The DRM Agent MUST include the following extension in the RightsEnablementRequest:

```
Extension() {
    AssetIndexList()          // Defined in Appendix E.2.11.2.6
}
```

The **`extensionType`** in the **`ExtensionsContainer()`** (see section 5.6.5) for this extension SHALL be set to '1'. The fields of the extension are defined as follows:

- **`AssetIndexList`** – list of BCRO Asset Indexes that indicates assets to be enabled

- If any unexpected exception occurs, the recovery from the exception SHALL be handled as a part of transaction that induced BCRO Asset Enablement.

6.12 SRM Utilities

The protocols specified in this section provide necessary functions that are used for the Rights Move and Local Rights Consumption.

6.12.1 Handle List Query

To read Rights from an SRM, the DRM Agent has to be aware of the identifier of the Rights in the SRM (i.e. Handle). The Handle List Query Message processing is used to read a Handle List from the SRM as illustrated in Figure 31.

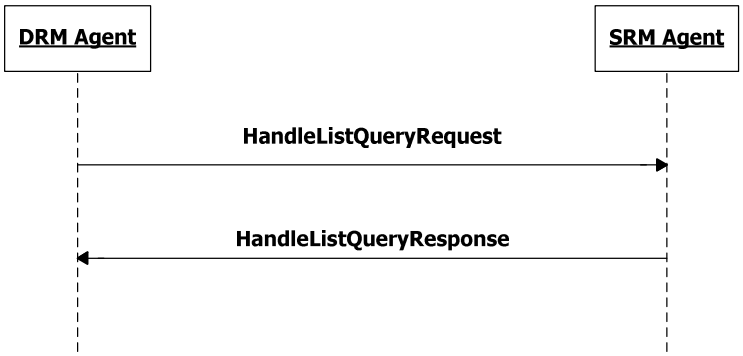

Figure 31: Sequence Diagram – Handle List Query

6.12.1.1 Description of Messages

The DRM Agent sends the HandleListQueryRequest to read the Handle List from the SRM. The field of the request is defined in Table 101.

Table 101: Fields of HandleListQueryRequest

Fields	Protection Requirement	Description
Hash Of AssetID List	No	List of H(AssetID)
Handle List Length	No	Maximum Handle List length in bytes that the DRM Agent can process. If this value is non-zero, the SRM Agent MUST send a Handle List shorter than or equal to the Handle List Length value. If a Handle List is longer than the Handle List Length, the SRM Agent divides the Handle List into several chunks.

The H(AssetID) in the HandleListQueryRequest is the identification of a DRM Content. The DRM Content can be associated with one or multiple Rights. The SRM Agent generates and returns a Handle List of Rights that are associated with the DRM Content.

Upon receiving the HandleListQueryRequest, the SRM Agent performs the following procedure:

- If the number of H(AssetID)'s in the *Hash Of AssetID List* is zero, the SRM Agent generates a Handle List of all enabled Rights in the SRM. If there are no Handles stored in the SRM, the SRM returns *Status* = *Success* and an empty *Handle List* in the response message.
- Otherwise, the SRM Agent generates a Handle List of enabled Rights in the SRM that are associated with the requested DRM Content identified by the H(AssetID)'s in the request. If there are no Handles stored in the SRM associated with the *Hash Of AssetID List*, the SRM returns *Status* = *Success* and an empty *Handle List* in the response message.

The SRM Agent sends the HandleListQueryResponse to carry the result of the action. The fields of the response are defined in Table 102. If the number of H(AssetID)s in the HandleListQueryRequest is more than the *Max Number Of AssetIDs* specified in the SrmHelloResponse, the SRM Agent MUST return the error - *AssetID List Too Long*.

Table 102: Fields of HandleListQueryResponse

Fields	Protection Requirement	Description
Status	No	The result of processing the HandleListQueryRequest message. The *Status* values are specified in Table 103. If *Status* contains any error, only this field is present in the HandleListQueryResponse.
Handle List	No	This field contains a Handle List or a chunk of it if a Handle List has been divided into several chunks. If no Handles exist in the SRM associated with the *Hash Of AssetID List* in the request, this field will contain no Handles.
Continuation Flag	No	It is assumed that a Handle List is divided into several chunks. '0': The *Handle List* in this response is the last chunk of the whole Handle List, or the Handle List is not divided into chunks '1': A Handle List has been divided into several chunks. The *Handle List* in this response is a chunk of the whole Handle List, and there are subsequent chunks.

Table 103: Status of Handle List Query Message

Status Value	Description
Success	The request was successfully processed.
AssetID List Too Long	The number of H(AssetID)s in the request exceeds the *Maximum Number Of AssetIDs*.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

If the *Continuation Flag* contains the value '1' and *Status* is *Success*, the DRM Agent SHOULD send the HandleListQueryRequest (containing same field values as the previous request) again in order to read the next chunk. The DRM Agent repeats the HandleListQueryRequest until the response contains the value '0' in the *Continuation Flag* field. If the DRM Agent receives excessive number of chunks, then the DRM Agent may not have an enough buffer to process all chunks. Then the DRM Agent can abort the repetition of the Handle List Query Message processing. If the *Continuation Flag* contains the value '0' and no errors or exceptions (*Status* = *Success*), the HandleListQueryRequest message processing is completed.

If the DRM Agent sends a different message or sends the HandleListQueryRequest with different field values than the previous values, then the SRM Agent resets the operation (i.e. the SRM Agent returns the Handle List from the first chunk again).

6.12.1.2 Format of Messages

The message format (**`MessageBody`**) of the HandleListQueryRequest is specified as follows. The **`messageType`** is set to '0' and the message is not protected by an HMAC.

```
MessageBody() {
    nbrOfAssetId                    8   uimsbf
    for ( i = 0 ; i < nbrOfAssetId ; i++ ) {
        HashOfAssetId()                 // Defined in Appendix E.3
    }
    handleListLength                16  uimsbf
}
```

The fields are defined as follows:

- **nbrOfAssetId** – Number of **HashOfAssetId** in *Hash Of AssetId List* field in Table 101
- **HashOfAssetId** – Hash of **AssetId**
- **handleListLength** – *Handle List Length* field in Table 101

The message format (**MessageBody**) of the HandleListQueryResponse is specified as follows. The **messageType** is set to '1' and the message is not protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        HandleList()                // Defined in Appendix E.4
        continuationFlag            1   bslbf
        rfu                         7   bslbf
    }
}
```

The fields are defined as follows:

- **status** - *Status* field in Table 102
- **HandleList** – *Handle List* field in Table 102
- **continuationFlag** – *Continuation Flag* field in Table 102

6.12.1.3 Exception Handling

There may be an unexpected exception during the Handle List Query Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response or finds an error by referring to the *Status*, then the DRM Agent regards it as an exception and terminates the Handle List Query Message processing. The User may be informed of the exception.

6.12.2 Rights Information Query

The DRM Agent requests the SRM Agent to read Rights Information including the Rights Meta Data, Rights Object Container, and State Information from the SRM as illustrated in Figure 32. The Rights Information does not include REK.

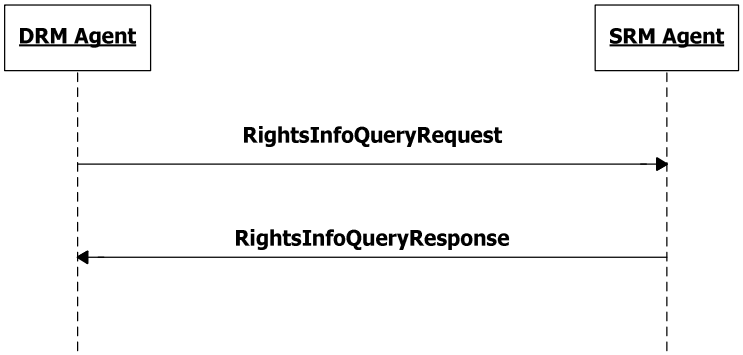

Figure 32: Sequence Diagram – Rights Information Query

6.12.2.1 Description of Messages

The DRM Agent sends the RightsInfoQueryRequest to read the Rights Information (Rights Meta Data, Rights Object Container, and State Information) without the REK from the SRM. The fields of the request are defined in Table 104.

Table 104: Fields of RightsInfoQueryRequest

Fields	Protection Requirement	Description
Handle	Integrity	This identifies Rights whose Rights Information will be transferred from the SRM to the Device. Refer to section 5.1.3.

Upon receiving the RightsInfoQueryRequest, the SRM Agent MUST perform the following procedure:

1. Verify the integrity of the request fields
2. Find Rights corresponding to the *Handle*

The SRM Agent sends the RightsInfoQueryResponse to carry the result of the procedure. The fields of the response are defined in Table 105.

Table 105: Fields of RightsInfoQueryResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the RightsInfoQueryRequest message. The *Status* values are specified in Table 106. If *Status* contains any error, only this field is present in the RightsInfoQueryResponse.
Rights Meta Data	Integrity	Refer to section 5.1.1.1
Rights Object Container	Integrity	Refer to section 5.1.1.2
Rights Information	Integrity	Refer to section 5.1.6.

Table 106: Status of Rights Information Query Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Handle Not Found	The Handle in the request does not exist in the SRM.

Status Value	Description
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response.

If no errors or exceptions (*Status* = *Success*), the Rights Information Query Message processing is completed.

6.12.2.2 Format of Messages

The message format (**`MessageBody`**) of the RightsInfoQueryRequest is specified as follows. The **`messageType`** is set to'0' and the message is protected by an HMAC.

```
MessageBody() {
    Handle()                        // Defined in Appendix E.2.4
}
```

The fields are defined as follows:

- **`Handle`** –*Handle* field in Table 104

The message format (**`MessageBody`**) of the RightsInfoQueryResponse is specified as follows. The **`messageType`** is set to '1' and the message is protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        RightsInformation()         // Defined in Appendix E.2.5.4
    }
}
```

The fields are defined as follows:

- **`RightsInformation`** – *Rights Meta Data*, *Rights Object Container*, *State Information* fields in Table 105
- **`Status`** - *Status* field in Table 105

6.12.2.3 Exception Handling

There may be an unexpected exception during the Rights Information Query Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response, finds an error by referring to the *Status* (except *Handle Not Found*), or fails to verify the integrity of fields, then the DRM Agent regards it as an exception and terminates communication with the SRM Agent. The User may be informed of the exception.

6.12.3 Rights Information List Query

The DRM Agent MAY request the SRM Agent to read the Rights Information List as illustrated in Figure 33.

The User may need to know the Rights Information before he/she can decide which Rights to retrieve. The Rights Information List Query message is used to read lists of Rights information from the SRM. By using this message, the DRM Agent SHOULD get the latest list of Handles with Rights Information from the SRM Agent before the Movement of Rights or Local Rights Consumption. The Rights Information List Query message is OPTIONAL for the SRMs.

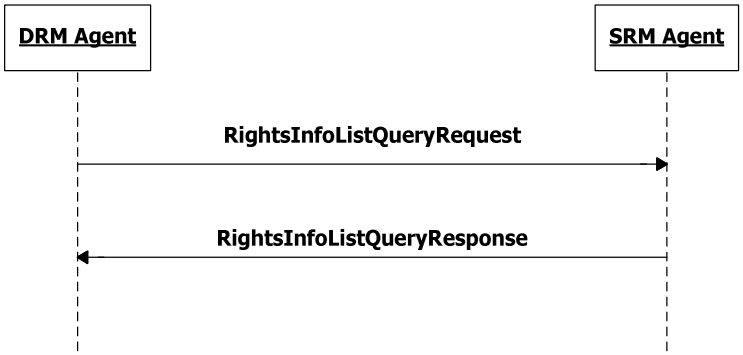

Figure 33: Sequence Diagram – Rights Information List Query

6.12.3.1 Description of Messages

The DRM Agent sends the RightsInfoListQueryRequest to read the Rights Information List from the SRM. The field of the request is defined in Table 107.

Table 107: Fields of RightsInfoListQueryRequest

Fields	Protection Requirement	Description
Hash Of AssetID List	No	List of H(AssetID)
Rights Information List Length	No	Maximum Rights Information List length in bytes that the DRM Agent can process. If this value is non-zero, the SRM Agent MUST send a Rights Information List shorter than or equal to the Rights Information List Length value. If a Rights Information List is longer than the Rights Information List Length, the SRM Agent divides the Rights Information List into several chunks.

The H(AssetID) in the RightsInfoListQueryRequest is the identification of a DRM Content. The DRM Content can be associated with one or multiple Rights. The SRM Agent generates and returns a Rights Information List that is associated with the DRM Content.

Upon receiving the RightsInfoListQueryRequest, the SRM Agent performs the following actions:

- If the number of H(AssetID)s in the *Hash Of AssetID List* is zero, the SRM Agent generates a Rights Information List of all enabled Rights in the SRM.
- Otherwise, the SRM Agent generates a Rights Information List of enabled Rights in the SRM that are associated with the requested DRM Contents identified by the H(AssetID)s in the request.

The SRM Agent sends the RightsInfoListQueryResponse to carry the result of the action. The fields of the response are defined in Table 108. If the number of H(AssetID)s in the RightsInfoListQueryRequest is more than the *Max Number Of AssetIDs* specified in the SrmHelloResponse, the SRM Agent MUST return the error - *AssetID List Too Long*.

Table 108: Fields of RightsInfoListQueryResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the RightsInfoListQueryRequest message. The *Status* values are specified in Table 109. If *Status* contains any error, only this field is present in the

Fields	Protection Requirement	Description
		RightsInfoListQueryResponse.
Rights Information List	Integrity	This field contains a Rights Information List or a chunk of it if the Rights Information List has been divided into several chunks.
Continuation Flag	Integrity	It is assumed that a Rights Information List is divided into several chunks. '0': The Rights Information List in this response is the last chunk of the whole Rights Information List, or the Rights Information List is not divided into chunks. '1': A Rights Information List has been divided into several chunks. The *Rights Information List* in this response is a chunk of the whole Rights Information List, and there are subsequent chunks.

Table 109: Status of Rights Information List Query Message

Status Value	Description
Success	The request was successfully processed.
AssetID List Too Long	The number of H(AssetID)s in the request exceeds the *Maximum Number Of AssetIDs*.
Request Not Supported	This request is not supported by the SRM Agent.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response.

If the Continuation Flag contains the value '1' and *Status* is *Success*, the DRM Agent SHOULD send the RightsInfoListQueryRequest again, with the same field values, in order to read the next chunk. The DRM Agent repeats the RightsInfoListQueryRequest until the response contains the value '0' in the *Continuation Flag* field. If the *Continuation Flag* contains the value '0' and no errors or exceptions (*Status* = *Success*), the RightsInfoListQueryRequest message processing is completed.

If the DRM Agent sends a different message or sends the RightsInfoListQueryRequest with different field values than the previous values, then the SRM Agent resets the operation (i.e. the SRM Agent returns the Rights Information List from the first chunk again).

If a Rights Information List is divided into several chunks, where a chunk will contain a portion of the complete list, the DRM Agent MUST concatenate all chunks in sequence from the SRM Agent in order to receive the complete Rights Information List.

6.12.3.2 Format of Messages

The message format (**`MessageBody`**) of the RightsInfoListQueryRequest is specified as follows. The **`messageType`** is set to '0' and the message is not protected by an HMAC.

```
MessageBody() {
    nbrOfAssetId                    8   uimsbf
    for ( i = 0 ; i < nbrOfAssetId ; i++ ) {
        HashOfAssetId()             // Defined in Appendix E.3
    }
    rightsInfoListLength            16  uimsbf
}
```

The fields are defined as follows:

- **`nbrOfAssetId`** – Number of **`HashOfAssetId`** in *Hash Of AssetId List* field in Table 107
- **`HashOfAssetId`** – Hash of **`AssetId`**
- **`rightsInfoListLength`** – *Rights Information List Length* field in Table 107

The message format (**`MessageBody`**) of the RightsInfoListQueryResponse is specified as follows. The **`messageType`** is set to ’0’ and the message is protected by an HMAC.

```
MessageBody() {
    Status()                            // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        RightsInformationList()         // Defined in Appendix E.2.6
        continuationFlag            1   bslbf
        rfu                         7   bslbf
    }
}
```

The fields are defined as follows:

- **`Status`** - *Status* field in Table 108
- **`RightsInformationList`** – *Rights Information List* field in Table 108
- **`continuationFlag`** – *Continuation Flag* field in Table 108

6.12.3.3 Exception Handling

There may be an unexpected exception during the Rights Information List Query Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response or finds an error by referring to the *Status*, then the DRM Agent regards it as an exception and terminates the Rights Information List Query Message processing. The User may be informed of the exception.

6.12.4 Handle Removal

The DRM Agent requests the SRM Agent to remove a Handle from the SRM as illustrated in Figure 34 when its corresponding Rights do not exist in the SRM.

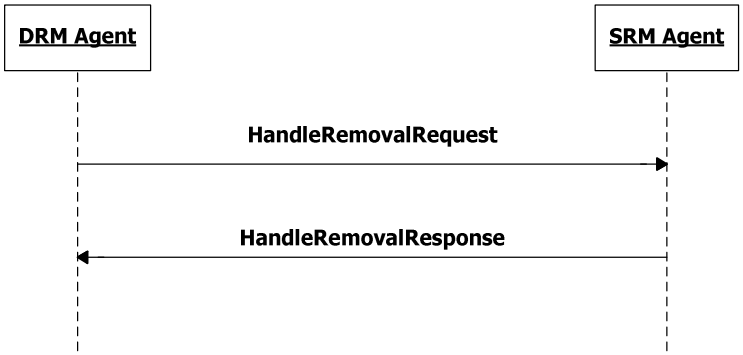

Figure 34: Sequence Diagram – Handle Removal

6.12.4.1 Description of Messages

The DRM Agent sends the HandleRemovalRequest to remove the Handle from the SRM. The fields of the request are defined in Table 110.

Table 110: Fields of HandleRemovalRequest

Fields	Protection Requirement	Description
Handle	Integrity & Confidentiality	*Handle* that will be removed from the SRM. Refer to section 5.1.3.

Upon receiving the HandleRemovalRequest, the SRM Agent MUST perform the following procedure:

1. Verify the integrity of the fields
2. Decrypt the *Handle* with the Session Key
3. Remove the matched *Handle* in the SRM

If Rights corresponding to the *Handle* already exist in the SRM, then the SRM Agent returns *Handle Not Removed* in the *Status*.

The SRM Agent sends the HandleRemovalResponse to carry the result of the procedure. The fields of the response are defined in Table 111.

Table 111: Fields of HandleRemovalResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the HandleRemovalRequest message. The *Status* values are specified in Table 112.

Table 112: Status of Handle Removal Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Handle Not Found	The SRM Agent cannot find the matched Handle.
Handle Not Removed	The SRM Agent cannot remove the *Handle* because Rights corresponding to the *Handle* already exist in the SRM.

Status Value	Description
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response.

If no errors or exceptions (*Status* = *Success*), the Handle Removal Message processing is completed.

6.12.4.2 Format of Messages

The message format (**`MessageBody`**) of the HandleRemovalRequest is specified as follows. The **`messageType`** is set to ’0’ and the message is protected by an HMAC.

```
MessageBody() {
    EncryptedHandle()              // Defined in Appendix E.2.8
}
```

The fields are defined as follows:

- **`EncryptedHandle`** – *Handle* field in Table 110 encrypted with the current Session Key (SK)

The message format (**`MessageBody`**) of the HandleRemovalResponse is specified as follows. The **`messageType`** is set to ‘1’ and the message is protected by an HMAC.

```
MessageBody() {
    Status()                       // Defined in Appendix E.2.2
}
```

The field is defined as follows:

- **`Status`** - *Status* field in Table 111

6.12.4.3 Exception Handling

There may be an unexpected exception during the Handle Removal Message processing as specified in section 5.5.1. The recovery from the exception is handled as a part of a Move transaction.

6.12.5 Rights Enablement

The DRM Agent requests the SRM Agent to enable Rights in the SRM using this function as illustrated in Figure 35.

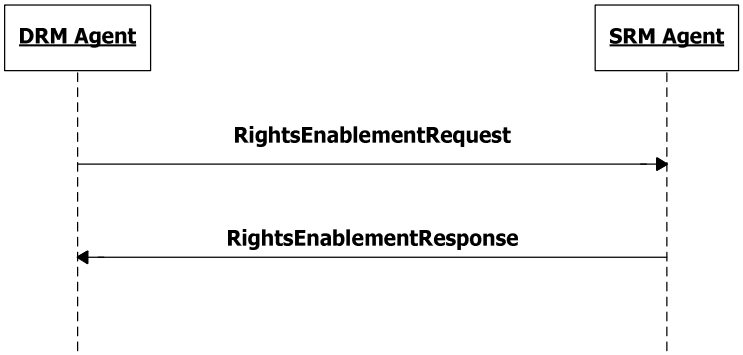

Figure 35: Sequence Diagram – Rights Enablement

6.12.5.1 Description of Messages

The DRM Agent sends the RightsEnablementRequest to enable the Rights in the SRM. The fields of the request are defined in Table 113.

Table 113: Fields of RightsEnablementRequest

Fields	Protection Requirement	Description
Handle	Integrity & Confidentiality	*Handle* identifies Rights that will be enabled in the SRM. Refer to section 5.1.3.
State Information	Integrity	New State Information that replaces the original State Information in the SRM. This field is OPTIONAL.

Upon receiving the RightsEnablementRequest, the SRM Agent MUST perform the following procedure:

1. Verify the integrity of the request fields
2. Decrypt the *Handle* with the Session Key
3. Enable the Rights corresponding to the *Handle*. If the *State Information* is present, overwrite the State Information of the found Rights in the SRM with the *State Information*. If the Rights are already enabled, then the SRM Agent returns *Success* in the *Status* field without executing this action.

The SRM Agent sends the RightsEnablementResponse to carry the result of the procedure. The fields of the response are defined in Table 114.

Table 114: Fields of RightsEnablementResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the RightsEnablementRequest message. The *Status* values are specified in Table 115.

Table 115: Status of Rights Enablement Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.

Status Value	Description
Handle Not Found	The Handle in the request does not exist in the SRM.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response.

If no errors or exceptions (*Status* = *Success*), the Rights Enablement Message processing is completed.

6.12.5.2 Format of Messages

The message format (**MessageBody**) of the RightsEnablementRequest is specified as follows. The **messageType** is set to'0' and the message is protected by an HMAC.

```
MessageBody() {
    stateInformationPresent  1      bslbf
    rfu                      7      bslbf

    EncryptedHandle()               // Defined in Appendix E.2.8
    if ( stateInformationPresent ) {
        StateInformation()          // Defined in Appendix E.2.5.3
    }
}
```

The fields are defined as follows:

- **EncryptedHandle** – *Handle* field in Table 113 encrypted with the current Session Key (SK)
- **stateInformationPresent** – if '1', then **StateInformation** is present in this message
- **StateInformation** – *State Information* field in Table 113

The message format (**MessageBody**) of the RightsEnablementResponse is specified as follows. The **messageType** is set to '1' and the message is protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
}
```

The field is defined as follows:

- **Status** - *Status* field in Table 114

6.12.5.3 Exception Handling

There may be unexpected exceptions during the Rights Enablement Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response, receives an error in the *Status* field, or fails to verify the message integrity, then the DRM Agent regards it as an exception.

The recovery from the exception is handled as a part of a Move or a Local Rights Consumption transaction.

6.12.6 Rights Removal

The DRM Agent requests the SRM Agent to remove Rights from the SRM using this function as illustrated in Figure 36.

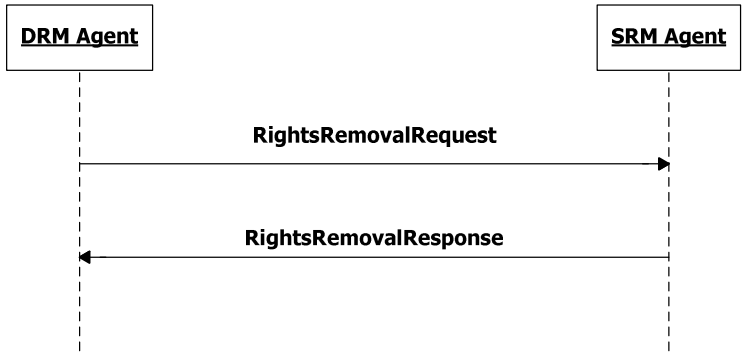

Figure 36: Sequence Diagram – Rights Removal

6.12.6.1 Description of Messages

The DRM Agent sends the RightsRemovalRequest to remove the Rights in the SRM. The fields of the request are defined in Table 116.

Table 116: Fields of RightsRemovalRequest

Fields	Protection Requirement	Description
Handle	Integrity & Confidentiality	*Handle* identifies Rights that will be removed in the SRM. Refer to section 5.1.3.

Upon receiving the RightsRemovalRequest, the SRM Agent performs the following procedure:

1. Verify the integrity of the request fields
2. Decrypt the *Handle* with the Session Key
3. Remove the Rights corresponding to the *Handle* and also remove the locally stored Handle

The SRM Agent sends the RightsRemovalResponse to carry the result of the procedure. The fields of the response are defined in Table 117.

Table 117: Fields of RightsRemovalResponse

Fields	Protection Requirement	Description
Status	Integrity	The result of processing the RightsRemovalRequest message. The *Status* values are specified in Table 118.

Table 118: Status of Rights Removal Message

Status Value	Description
Success	The request was successfully processed.
Field Integrity Verification Failed	The HMAC value of fields in the request did not match the HMAC value generated by the SRM Agent.
Handle Not Found	The Handle in the request does not exist in the SRM.
Parameter Failed	A field in the request has an invalid length or structure.

Status Value	Description
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

Upon receiving the response, the DRM Agent verifies the integrity of fields in the response.

If no errors or exceptions (*Status* = *Success*), the Rights Removal Message processing is completed.

6.12.6.2 Format of Messages

The message format (**`MessageBody`**) of the RightsRemovalRequest is specified as follows. The **`messageType`** is set to ’0’ and the message is protected by an HMAC.

```
MessageBody() {
    EncryptedHandle()              // Defined in Appendix E.2.8
}
```

The fields are defined as follows:

- **`EncryptedHandle`** – *Handle* field in Table 116 encrypted with the current Session Key (SK)

The message format (**`MessageBody`**) of the RightsRemovalResponse is specified as follows. The **`messageType`** is set to ‘1’ and the message is protected by an HMAC.

```
MessageBody() {
    Status()                       // Defined in Appendix E.2.2
}
```

The field is defined as follows:

- **`Status`** - *Status* field in Table 117

6.12.6.3 Exception Handling

There may be an unexpected exception during the Rights Removal Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response, receives an error in the *Status* field, or fails to verify the message integrity, then the DRM Agent regards it as an exception.

The recovery from the exception is handled as a part of a Move transaction.

6.12.7 Store RI Certificate Chain

The DRM Agent requests the SRM Agent to store Rights Issuer’s certificate chains in the SRM as illustrated in Figure 37. The DRM and SRM Agents MAY support these messages.

Figure 37: Sequence Diagram – Store RI Certificate Chain

6.12.7.1 Description of Messages

The DRM Agent sends the RICertificateStoreRequest to store an RI certificate chain in the SRM. The fields of the request are defined in Table 119.

Table 119: Fields of RICertificateStoreRequest

Fields	Protection Requirement	Description
RI ID	No	The hash of the Rights Issuer's public key in the RI Certificate (i.e. the hash of the complete DER-encoded subjectPublicKeyInfo component in the RI Certificate). The default hash algorithm is SHA-1.
RI Certificate Chain	No	Rights Issuer's certificate chain

Upon receiving the RICertificateStoreRequest, the SRM Agent stores the RI ID and certificate chain. If there already exists the RI certificate chain, this is overwritten with the certificate chain in the request.

The SRM Agent sends the RICertificateStoreResponse to carry the result of the action. The fields of the response are defined in Table 120.

Table 120: Fields of RICertificateStoreResponse

Fields	Protection Requirement	Description
Status	No	The result of processing the RICertificateStoreRequest message. The *Status* values are specified in Table 121.

Table 121: Status of RI Certificate Store Message

Status Value	Description
Success	The request was successfully processed.
Not Enough Space	The SRM does not have enough space to store the certificate chain.
Request Not Supported	RI Certificate Chain cannot be stored in the SRM.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

If no errors or exceptions (*Status* = *Success*), the RI Certificate Store Message processing is completed.

6.12.7.2 Format of Messages

The message format (**MessageBody**) of the RICertificateStoreRequest is specified as follows. The **messageType** is set to ’0’ and the message is not protected by an HMAC.

```
RiId() {
    OctetString8()                  // Defined in Appendix E.1
}

RiCertificateChain() {
    CertificateChain()              // Defined in Appendix E.1
}

MessageBody() {
    RiId()
    RiCertificateChain()
}
```

The fields are defined as follows:

- **RiId** – *RI ID* field in Table 119
- **RiCertificateChain** – *RI Certificate Chain* field in Table 119

The message format (**MessageBody**) of the RICertificateStoreResponse is specified as follows. The **messageType** is set to ‘1’ and the message is not protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
}
```

The field is defined as follows:

- **Status** - *Status* field in Table 120

6.12.7.3 Exception Handling

There may be an unexpected exception during the RI Certificate Store Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response, or finds an error by referring to the *Status* (except *Not Enough Space* and *Request Not Supported*), then the DRM Agent regards it as an exception and terminates communication by discarding any existing SAC context with the SRM Agent. The User MAY be informed of this exception.

6.12.8 Get RI Certificate Chain

The DRM Agent requests the SRM Agent to read Rights Issuer’s certificate chains from the SRM as illustrated in Figure 38. The DRM and SRM Agents MAY support these messages.

Figure 38: Sequence Diagram – Get RI Certificate Chain

6.12.8.1 Description of Messages

The DRM Agent sends the RICertificateQueryRequest to read an RI certificate chain from the SRM. The fields of the request are defined in Table 122.

Table 122: Fields of RICertificateQueryRequest

Fields	Protection Requirement	Description
RI ID	No	The hash of the Rights Issuer's public key in the RI Certificate (i.e. the hash of the complete DER-encoded subjectPublicKeyInfo component in the RI Certificate). The default hash algorithm is SHA-1.

Upon receiving the RICertificateQueryRequest, the SRM Agent reads the RI certificate chain identified by the RI ID from the SRM.

The SRM Agent sends the RICertificateQueryResponse to carry the result of the action. The fields of the response are defined in Table 123.

Table 123: Fields of RICertificateQueryResponse

Fields	Protection Requirement	Description
Status	No	The result of processing the RICertificateQueryRequest message. The *Status* values are specified in Table 124. If *Status* contains any error, only this field is present in the RICertificateQueryResponse.
RI Certificate Chain	No	Rights Issuer's certificate chain

Table 124: Status of RI Certificate Query Message

Status Value	Description
Success	The request was successfully processed.
RI Certificate Chain Not Found	The SRM Agent cannot find the matched RI certificate chain.
Request Not Supported	This request is not supported by the SRM Agent.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

If no errors or exceptions (*Status* = *Success*), the RI Certificate Query Message processing is completed.

6.12.8.2 Format of Messages

The message format (**MessageBody**) of the RICertificateQueryRequest is specified as follows. The **messageType** is set to ’0’ and the message is not protected by an HMAC.

```
RiId() {
    OctetString8()                  // Defined in Appendix E.1
}

MessageBody() {
    RiId()
}
```

The field is defined as follows:

- **RiId** – *RI ID* field in Table 122

The message format (**MessageBody**) of the RICertificateQueryResponse is specified as follows. The **messageType** is set to ‘1’ and the message is not protected by an HMAC.

```
RiCertificateChain() {
    CertificateChain()              // Defined in Appendix E.1
}

MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        RiCertificateChain()
    }
}
```

The fields are defined as follows:

- **Status** - *Status* field in Table 123
- **RiCertificateChain** – *RI Certificate Chain* field in Table 123

6.12.8.3 Exception Handling

There may be an unexpected exception during the RI Certificate Query Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response, or finds an error by referring to the *Status* (except *RI Certificate Chain Not Found* and *Request Not Supported*), then the DRM Agent regards it as an exception and terminates communication by discarding any existing SAC context with the SRM Agent. The User MAY be informed of this exception.

6.12.9 Remove RI Certificate Chain

The DRM Agent requests the SRM Agent to remove an RI certificate chain from the SRM as illustrated in Figure 39. The DRM and SRM Agents MAY support these messages.

Figure 39: Sequence Diagram – Remove RI Certificate Chain

6.12.9.1 Description of Messages

The DRM Agent sends the RICertificateRemovalRequest to remove the RI certificate chain from the SRM. The fields of the request are defined in Table 125.

Table 125: Fields of RICertificateRemovalRequest

Fields	Protection Requirement	Description
RI ID	No	The hash of the Rights Issuer's public key in the RI Certificate (i.e. the hash of the complete DER-encoded subjectPublicKeyInfo component in the RI Certificate). The default hash algorithm is SHA-1.

Upon receiving the RICertificateRemovalRequest, the SRM Agent finds the RI certificate chain identified by the *RI ID* and removes it.

The SRM Agent sends the RICertificateRemovalResponse to carry the result of the action. The fields of the response are defined in Table 126.

Table 126: Fields of RICertificateRemovalResponse

Fields	Protection Requirement	Description
Status	No	The result of processing the RICertificateRemovalRequest message. The *Status* values are specified in Table 127.

Table 127: Status of RI Certificate Removal Message

Status Value	Description
Success	The request was successfully processed.
RI Certificate Chain Not Found	The SRM Agent cannot find the matched RI certificate chain.
Request Not Supported	This request is not supported by the SRM Agent.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

If no errors or exceptions (*Status* = *Success*), the RI Certificate Removal Message processing is completed.

6.12.9.2 Format of Messages

The message format (**MessageBody**) of the RICertificateRemovalRequest is specified as follows. The **messageType** is set to ’0’ and the message is not protected by an HMAC.

```
RiId() {
    OctetString8()                  // Defined in Appendix E.1
}

MessageBody() {
    RiId()
}
```

The field is defined as follows:

- **RiId** – *RI ID* field in Table 125

The message format (**MessageBody**) of the RICertificateRemovalResponse is specified as follows. The **messageType** is set to ‘1’ and the message is not protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
}
```

The field is defined as follows:

- **Status** - *Status* field in Table 126

6.12.9.3 Exception Handling

There may be an unexpected exception during the RI Certificate Removal Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response, or finds an error by referring to the *Status* (except *RI Certificate Chain Not Found* and *Request Not Supported*), then the DRM Agent regards it as an exception and terminates communication by discarding any existing SAC context with the SRM Agent. The User MAY be informed of this exception.

6.12.10 Dynamic Code Page Query

The DRM Agent requests the SRM Agent to read the WBXML Dynamic Code Pages (see section 8.2.2 and 8.3.2) as illustrated in Figure 40. The SRM Agent MAY support these messages.

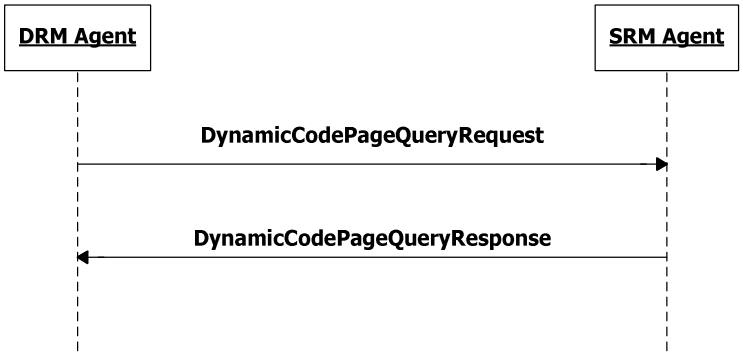

Figure 40: Sequence Diagram – Dynamic Code Page Query

6.12.10.1 Description of Messages

The DRM Agent sends the DynamicCodePageQueryRequest to read the WBXML Dynamic Code Pages from the SRM. The DynamicCodePageQueryRequest has no fields.

Upon receiving the DynamicCodePageQueryRequest, the SRM Agent reads the Dynamic Code Pages from its internal storage and the SRM Agent sends the DynamicCodePageQueryResponse to carry the result of the action. The fields of the response are defined in Table 128.

Table 128: Fields of DynamicCodePageQueryResponse

Fields	Protection Requirement	Description
Status	No	The result of processing the DynamicCodePageQueryRequest message. The *Status* values are specified in Table 129. If *Status* contains any error, only this field is present in the DynamicCodePageQueryResponse.
Attribute Code Page	No	The Dynamic Attribute Code Page
Tag Code Page	No	The Dynamic Tag Code Page

Table 129: Status of Dynamic Code Page Query Message

Status Value	Description
Success	The request was successfully processed.
Dynamic Code Pages Not Found	The WBXML Dynamic Code Pages do not yet exist on the SRM. The DRM Agent MAY create new Code Pages as required.
Request Not Supported	This request is not supported by the SRM Agent.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

If no errors or exceptions (*Status = Success* or *Status = Dynamic Code Pages Not Found*), the Dynamic Code Page Query Message processing is completed.

6.12.10.2 Format of Messages

The **MessageBody** of the DynamicCodePageQueryRequest is empty. The **messageType** is set to ’0’ and the message is not protected by an HMAC.

The message format (**MessageBody**) of the DynamicCodePageQueryResponse is specified as follows. The **messageType** is set to ‘1’ and the message is not protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
    if ( Status == 0 ) {
        AttributeCodePage()         // Defined in Appendix E.5.1
        TagCodePage()               // Defined in Appendix E.5.2
    }
}
```

The fields are defined as follows:

- **Status** - *Status* field in Table 128
- **AttributeCodePage** – *Attribute Code Page* field as defined in E.5.1
- **TagCodePage** – *Tag Code Page* field in section E.5.2

6.12.10.3 Exception Handling

There may be an unexpected exception during the Dynamic Code Page Query as specified in section 5.5.1. If the DRM Agent fails to receive the response, or finds an error by referring to the *Status* (except *Dynamic Code Pages Not Found* and *Request Not Supported*), then the DRM Agent regards it as an exception and terminates communication by discarding any existing SAC context with the SRM Agent. The User MAY be informed of this exception.

6.12.11 Dynamic Code Page Update

The DRM Agent requests the SRM Agent to store an updated WBXML Dynamic Code Page on the SRM as illustrated in Figure 41. The DRM and SRM Agents MAY support these messages.

Figure 41: Sequence Diagram – Dynamic Code Page Update

6.12.11.1 Description of Messages

The DRM Agent sends the DynamicCodePageUpdateRequest to store an updated set of Dynamic Code Pages in the SRM. The updated code pages SHALL replace any existing code pages. The fields of the request are defined in Table 130.

Table 130: Fields of DynamicCodePageUpdateRequest

Fields	Protection Requirement	Description
Attribute Code Page	No	The Dynamic Attribute Code Page
Tag Code Page	No	The Dynamic Tag Code Page

Upon receiving the DynamicCodePageUpdateRequest, the SRM Agent stores the updated code page(s). If the updated code page already exists in the SRM then it is overwritten.

The SRM Agent sends the DynamicCodePageUpdateResponse to carry the result of the action. The fields of the response are defined in Table 131.

Table 131: Fields of DynamicCodePageUpdateResponse

Fields	Protection Requirement	Description
Status	No	The result of processing the DynamicCodePageUpdateRequest message. The *Status* values are specified in Table 132

Table 132: Status of Dynamic Code Page Update Message

Status Value	Description
Success	The request was successfully processed.
Not Enough Space	The SRM does not have enough space to store the dynamic code tables
Request Not Supported	Dynamic Code Tables cannot be stored in the SRM.
Parameter Failed	A field in the request has an invalid length or structure.
Unexpected Request	This request was received out of sequence or is otherwise not allowed.
Unknown Error	Other errors

If no errors or exceptions (*Status* = *Success*), the Dynamic Code Page Update Message processing is completed.

6.12.11.2 Format of Messages

The message format (**`MessageBody`**) of the DynamicCodePageUpdateRequest is specified as follows. The **`messageType`** is set to '0' and the message is not protected by an HMAC.

```
MessageBody() {
    attributeCodePagePresent        1   bslbf
    tagCodePagePresent              1   bslbf
    rfu                             6   bslbf
    if ( attributeCodePagePresent ) {
        AttributeCodePage()         // Defined in Appendix E.5.1
    }
    if ( tagCodePagePresent ) {
        TagCodePage()               // Defined in Appendix E.5.2
    }
}
```

The fields are defined as follows:

- **`attributeCodePagePresent`** – indicates whether the **`AttributeCodePage`** is present.

- **tagCodePagePresent** – indicates whether the **TagCodePage** is present.
- **AttributeCodePage** – *Attribute Code Page* field as defined in E.5.1
- **TagCodePage** – *Tag Code Page* field in section E.5.2

The message format (**MessageBody**) of the DynamicCodePageUpdateResponse is specified as follows. The **messageType** is set to '1' and the message is not protected by an HMAC.

```
MessageBody() {
    Status()                        // Defined in Appendix E.2.2
}
```

The field is defined as follows:

- **Status** - *Status* field in Table 131

6.12.11.3 Exception Handling

There may be an unexpected exception during the Dynamic Code Page Update Message processing as specified in section 5.5.1. If the DRM Agent fails to receive the response, or finds an error by referring to the *Status* (except *Not Enough Space* and *Request Not Supported*), then the DRM Agent regards it as an exception and terminates communication by discarding any existing SAC context with the SRM Agent. The User MAY be informed of this exception.

7. ROAP Extension

These sections define the extension to DRM v2.0 ROAP protocol messages and SCE-ROAP protocol.

7.1 ROAP Trigger

This specification extends XML Schema for ROAP RO Acquisition Trigger for Direct Provisioning of Rights to the SRM procedure as shown in Figure 16. A ROAP RO Acquisition Trigger MAY be delivered from an RI to a DRM Agent to initiate the RO Acquisition protocol for Direct Provisioning of Rights to SRM. Rights Issuer has to indicate to the DRM Agent in host device that the Rights Object is to be downloaded and installed to the SRM Agent. The ROAP RO Acquisition Trigger SHALL include at least one <trustAnchorAndsrmIDPair> element for Direct Provisioning of Rights to the SRM procedure. In case of Direct Provisioning of Rights to the SRM, a DRM Agent on receiving ROAP RO Acquisition Trigger with <trustAnchorAndsrmIDPair> element MUST interact with a SRM that have identifier matching with <srmID> element under the trust anchor identified trust model, to extract the necessary information used to generate the ROAP RO Request message, and consequently send the built ROAP RO Request message to the Rights Issuer.

The schema below depicts the additional element to the RO Acquisition Trigger schema.

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

The <trustAnchorAndsrmIDPair> element identifies the SRM under a trust model to which the Rights Object is to be downloaded and installed. If the RO Acquisition Trigger includes a <trustAnchorAndsrmIDPair> element, the DRM Agent SHALL initiate the Direct Provisioning of Rights to the SRM procedure as specified in Section 6.8 after receiving the RO Acquisition Trigger. If an SRM support multiple trust models, the RI sends several <trustAnchorAndsrmIDPair> elements.

The following schema fragment defines the **trustAnchorAndsrmIDPair** type.

```
<complex Type name="trustAnchorAndsrmIdentifierPair">
 <sequence>
   <element name="trustAnchor" type="roap:Identifier"/>
   <element name="srmID" type="roap:Identifier"/>
 </sequence>
</complexType>
```

The <trustAnchor> element identifies the trust model. The only identifier currently defined is the hash of the root CA's public key info, as it appears in the certificate (i.e. the hash of the complete DER-encoded `subjectPublicKeyInfo` component in the root CA's certificate). The default hash algorithm is SHA-1.

The <srmID> element identifies the SRM. The only identifier currently defined is the hash of the SRM's public key info, as it appears in the certificate (i.e. the hash of the complete DER-encoded `subjectPublicKeyInfo` component in the SRM's certificate under the Trust Anchor). The default hash algorithm is SHA-1.

7.2 RO Request

This specification defines a Trust Anchor extension to the ROAP RO Request message. When the RO Request is used for Direct Provisioning of Rights to the SRM as shown in Figure 16, the Trust Anchor extension MUST be included and marked as critical. This extension identifies the Trust Anchor used by the SRM Agent for acquiring rights, so that the RI can verify SRM Agent's signature on the RO Request. The value of this extension MUST match one of the Trust Anchors given by the <trustAnchorAndsrmIDPair> element(s) in the RO Acquisition trigger.

The following schema fragment defines the Trust Anchor extension:

```
<complexType name="TrustAnchor">
  <complexContent>
    <extension base="roap:Extension">
      <element name="ta" type="roap:Identifier"/>
    </extension>
  </complexContent>
</complexType>
```

7.3 Extension to SCE ROAP-ROUpgrade Trigger

The RO Upgrade Trigger format is extension of SCE ROAP-ROUpgradeTrigger. In this trigger, the element of *assetID* SHALL be carried under the TriggerInformation element as specified in [SCE-GEN].

```
<complexType name="TriggerInformation">
 <sequence>
          <element name="assetID" type="ID" />
          <any minOccurs="0" maxOccurs="unbounded" processContents="lax" namespace="##any"/>
 </sequence>
</complexType>
```

The *assetID* specifies the content that has associated RO needed to be updated.

8. Compact Encoding of Rights

This section specifies the compact encoding of the SRM 1.0 Rights Object Container (<**oma-dd:roContainer**>). This encoding is used if the Rights Object Container `roFormat` indicates WBXML encoding.

WBXML 1.3 [WBXML] is a simple method that allows compacting XML documents in a loss-less manner. A WBXML decoder processes a WBXML encoded document by interpreting it byte-by-byte. Some bytes represent decoding instructions, some represent XML element start tags, attribute names or attribute values. The decoding process is stateful. The decoder maintains one global state, which determines whether it is processing elements, or attributes. Within each state, the decoder maintains an independent notion of a selected code page.

8.1 WBXML Encoding Rules

The following rules MUST be followed when WBXML encoding the <**roContainer**>:

- WBXML version 1.3 MUST be used (encoded as u_int8 value 0x03)
- The public identifier value "-//OMA//SRM 1.0//EN" MUST be used (encoded as mb_u_int32 value of 0x14). This document type identifier is registered by OMNA.
- The character set MUST be UTF-8 (encoded as mb_u_int32 value 0x6A).
- The string table MAY be used to specify string values of literal BCAST Tokens. The string table SHOULD NOT be used to encode WBXML strings.
- All strings SHOULD be encoded inline.

8.2 Attribute Code Pages

8.2.1 Fixed Attribute Code Page

Attribute code page 0 in the context of the public identifier "-//OMA//SRM 1.0//EN" is a fixed code page. This holds attribute names and attribute values that correspond to the Rights Objects used in SRM 1.0, DRM 2.0, DRM 2.1 and BCAST 1.0. This code table is fixed and future versions of SRM will not add additional values to this code table.

Table 133: Fixed WBXML Attribute Code Page – Attribute Names

Attribute Name	WBXML Attribute Code	Comment
GLOBAL TOKENS	00 – 04	
xmlns:o-ex	05	
xmlns:o-dd	06	
xmlns:ds	07	
xmlns:oma-dd	08	
xmlns:xenc	09	
o-ex:id	0A	
o-ex:idref	0B	
Algorithm	0C	
URI	0D	
oma-dd:onExpiredURL	0E	
oma-dd:timer	0F	
oma-dd:mode	10	
oma-dd:timed	11	
oma-dd:contentAccessGranted	12	
oma-dd:token-timed-count-timer	13	

Attribute Name	WBXML Attribute Code	Comment
xmlns:roap	14	
	15 – 3F	Not Used
	40 – 44	GLOBAL TOKENS
	45 – 7F	Not Used

Table 134: Fixed WBXML Attribute Code Page – Attribute Values

Attribute Value	WBXML Attribute Value Code
GLOBAL TOKENS	80 - 84
http://odrl.net/1.1/ODRL-EX	85
http://odrl.net/1.1/ODRL-DD	86
http://www.openmobilealliance.com/oma-dd	87
http://www.w3.org/2000/09/xmldsig#	88
http://www.w3.org/2001/04/xmlenc#	89
http://www.w3.org/2000/09/xmldsig#sha1	8A
http://www.w3.org/2001/04/xmlenc#kw-aes128	8B
http://www.w3.org/2001/10/xml-exc-c14n#	8C
http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-1#rsa-pss-default	8D
#K_MAC_and_K_REK	8E
move	8F
copy	90
true	91
false	92
urn:oma:bac:dldrm:roap-1.0	93
UNUSED	94 - BF
GLOBAL TOKENS	C0 – C4
UNUSED	C5 – FF

8.2.2 Dynamic Attribute Code Pages

Attribute code page 1 in the context of the public identifier "-//OMA//SRM 1.0//EN" is a dynamic code page. The attribute name codes and attribute value codes are not defined in this specification. The dynamic attribute code page is defined to allow forward compatibility in WBXML encoding. The values within the dynamic attribute code page are stored on the SRM. The dynamic code page is unique per SRM. Normally a new SRM will have an empty dynamic attribute code page.

As defined in section 6.12.10, DRM Agents MAY retrieve the SRM's dynamic code page from the SRM. DRM Agents MUST NOT delete attribute codes from the SRM's dynamic code page.

During the process of compacting an XML Document before transferring it to the SRM, the DRM Agent MAY add new attribute code values and attribute name codes to the dynamic code page. DRM Agents SHOULD add new attribute codes if the XML Document to be compacted contains any attributes or attribute values that do not exist in either code page 0 or the existing dynamic code page.

8.2.3 Reserved Attribute Code Pages

All attribute code pages 2 to 127 in the context of the public identifier "-//OMA//SRM 1.0//EN" are reserved for future use by OMA.

8.3 Tag Code Pages

8.3.1 Rights Object Container

The element tag code page defined in this section is to be used together with tag code page 1 and attribute code pages 0 and 1 to encode and decode the Rights Object Container's **<roContainer>** element.

Table 135: Fixed WBXML Tag Code Page

Tag Name	WBXML Attribute Code			
	No content, No attributes	Content, No attributes	No Content, Attributes	Content, Attributes
rights	05	45	85	C5
o-ex:context	06	46	86	C6
o-ex:agreement	07	47	87	C7
o-ex:asset	08	48	88	C8
o-ex:inherit	09	49	89	C9
o-ex:permission	0A	4A	8A	CA
o-ex:requirement	0B	4B	8B	CB
o-ex:constraint	0C	4C	8C	CC
o-ex:digest	0D	4D	8D	CD
o-dd:version	0E	4E	8E	CE
o-dd:uid	0F	4F	8F	CF
o-dd:play	10	50	90	D0
o-dd:display	11	51	91	D1
o-dd:execute	12	52	92	D2
o-dd:print	13	53	93	D3
o-dd:export	14	54	94	D4
o-dd:move	15	55	95	D5
o-dd:save	16	56	96	D6
o-dd:tracked	17	57	97	D7
o-dd:count	18	58	98	D8
o-dd:datetime	19	59	99	D9
o-dd:start	1A	5A	9A	DA
o-dd:end	1B	5B	9B	DB
o-dd:interval	1C	5C	9C	DC
o-dd:accumulated	1D	5D	9D	DD
o-dd:individual	1E	5E	9E	DE
oma-dd:timed-count	1F	5F	9F	DF
oma-dd:system	20	60	A0	E0
oma-dd:access	21	61	A1	E1
oma-dd:token-based	22	62	A2	E2
oma-dd:token-constraint-count	23	63	A3	E3
oma-dd:token-constraint-timed-count	24	64	A4	E4
oma-dd:token-accumulated	25	65	A5	E5
oma-dd:token-unit	26	66	A6	E6
oma-dd:token-consumed	27	67	A7	E7
oma-dd:roContainer	28	68	A8	E8

Tag Name	WBXML Attribute Code			
	No content, No attributes	Content, No attributes	No Content, Attributes	Content, Attributes
xenc:EncyptedKey	29	69	A9	E9
xenc:EncryptionMethod	2A	6A	AA	EA
xenc:CipherData	2B	6B	AB	EB
xenc:CipherValue	2C	6C	AC	EC
ds:DigestMethod	2D	6D	AD	ED
ds:DigestValue	2E	6E	AE	EE
ds:KeyInfo	2F	6F	AF	EF
ds:RetrievalMethod	30	70	B0	F0
ds:SignedInfo	31	71	B1	F1
ds:CanonicalizationMethod	32	72	B2	F2
ds:SignatureMethod	33	73	B3	F3
ds:Reference	34	74	B4	F4
ds:Transforms	35	75	B5	F5
ds:Transform	36	76	B6	F6
ds:SignatureValue	37	77	B7	F7
roap:X509SPKIHash	38	78	B8	F8
hash	39	79	B9	F9
ds:Signature	3A	7A	BA	FA
oma-dd:srmPing	3B	7B	BB	FB
oma-dd:checkInterval	3C	7C	BC	FC

In the context of the public identifier "-//OMA//SRM 1.0//EN", within tag code page 0, tag codes from the range 3D-3F, 7D-7F, BD-BF and FD-FF are reserved for future use by OMA.

8.3.2 Dynamic Tag Code Page

Tag code page 1 in the context of the public identifier "-//OMA//SRM 1.0//EN" is a dynamic code page. The tag name codes are not defined in this specification. The dynamic tag code page is defined to allow forward compatibility in WBXML encoding. The values within the dynamic tag code page are stored on the SRM. The dynamic code page is unique per SRM. Normally a new SRM will have an empty dynamic tag code page.

As defined in section 6.12.10, DRM Agents MAY retrieve the SRM's dynamic tag code page from the SRM. DRM Agents MUST NOT delete tag-name codes from the SRM's dynamic code page.

During the process of compacting an XML document before transferring it to the SRM, the DRM Agent MAY add new tag-name codes to the dynamic code page. DRM Agents SHOULD add new tag name codes if the XML document to be compacted contains any tags that do not exist in either code page 0 or the existing dynamic code page.

8.3.3 Reserved Tag Code Pages

All tag code pages 2 to 127 in the context of the public identifier "-//OMA//SRM 1.0//EN" are reserved for future use by OMA.

8.4 Processing

8.4.1 Device (DRM Agent)

DRM Agents MUST support WBXML encoding of the **<roContainer>** element as defined in section E.2.5.2. DRM Agents SHOULD be able to generate and extend dynamic code pages if they are supported by the target SRM. DRM Agents

SHOULD be able to encode tags, and attributes that do not have a well known token-code using the WBXML LITERAL token.

DRM Agents MUST support WBXML decoding of the **<roContainer>** element as defined in section E.2.5.2.

DRM Agents MAY support updating of SRM dynamic code tables.

8.4.2 SRM (SRM Agent)

SRM Agents MAY support storage of the dynamic attribute code page and dynamic tag code page.

SRM Agents do not need to support WBXML encoding or decoding.

8.4.3 Rights Issuers

As the WBXML encoding of Rights Object Containers is supported by DRM Agents, Rights Issuers do not need to support WBXML encoding of Rights Object Containers.

8.5 Data Representation

8.5.1 Binary Data Representation

The WBXML OPAQUE token provides a method to encode raw binary data. DRM Agents MUST use the WBXML OPAQUE token to represent whitespace in the XML.

8.5.2 base64Binary Representation

Some elements in the Rights Object Container hold base64Binary data. All base64Binary data within the **<rights>** element MUST be encoded in the WBXML form directly as a literal string. All base64Binary data within the **<signature>** element MUST be base64 decoded prior to WBXML encoding and then encoded using the OPAQUE token.

8.6 Normal Processing and Transcoding

After a DRM Agent receives a WBXML encoded **<roContainer>** from an SRM, that DRM Agent MUST decode the message into XML format before any other processing is applied. Before a DRM Agent Moves a Rights Object Container to an SRM, the DRM Agent MUST canonicalise contents of the **<rights>** element of XML using Exclusive Canonicalisation without comments, and MAY encode the **<roContainer>** element using WBXML. The criterion by which a DRM Agent determines whether it should WBXML encode an **<roContainer>** is outside the scope of this specification. It is anticipated that individual SRM form factors will have different recommendations. In general, it is RECOMMENDED that DRM Agents SHOULD WBXML encode all **<roContainer>** elements larger than 2kB.

If the SRM does not support storage of dynamic code pages then the DRM Agent MUST use only the fixed code pages. If the **<roContainer>** contains any additional tags, attributes or attribute values, these SHOULD be encoded using the WBXML LITERAL token.

The WBXML SWITCH token SHALL be used to switch between the fixed code pages and dynamic code pages.

The normal process for a DRM Agent to encode a Rights Object Container is:

- Construct the **<roContainer>** as XML
  - DRM Agent MUST canonicalise contents of the **<rights>** element of XML using Exclusive Canonicalisation without comments.
- Encode the **<roContainer>** using as WBXML:
  - Initially attempt to use only the fixed attribute code page and fixed tag code page.
  - If the SRM Agent indicated support for dynamic code pages in the SrmHelloResponse message then:

        - If during the encoding any unknown attributes, attribute values or tags are found, then DRM Agents SHOULD retrieve the dynamic code pages from the SRM, before continuing processing.
        - Continue the encoding making use of the additional attributes, attribute values, and tags that are specified in the dynamic code pages.
        - If the dynamic code pages do not contain the necessary attributes, then the DRM Agent SHOULD add new attributes codes, attribute values and tags as required to the relevant dynamic tag code page.
        - If updates were made to a dynamic code table then the DRM Agent MUST store the updated code table on the SRM. If the update to the dynamic code pages on the SRM fails for any reason, the DRM Agent SHOULD discard the WBXML encoded **<roContainer>**. The DRM Agent MAY re-start the processing as if the SRM Agent does not support dynamic code pages; or it MAY use plain XML to represent the **<roContainer>**.
    - If the SRM Agent does not support dynamic code pages:
        - Continue the encoding making use of the WBXML LITERAL token to encode unknown attributes and tags.

The normal process for a DRM Agent to decode a WBXML encoded **<roContainer>** is:

- Attempt to decode the WBXML encoded **<roContainer>** to XML:
    - Initially attempt to decode using only the fixed attribute code page and fixed tag code page.
    - If during the decoding any unknown application tokens are discovered, then if the SRM indicated support for dynamic code pages in the SrmHelloResponse message then retrieve the dynamic code pages from the SRM.
    - Continue the decoding by making use of the additional application tokens that are specified in the dynamic code pages.
    - If an application token is not specified in either the fixed code pages or dynamic code pages, then a critical error has occurred. The DRM Agent SHOULD delete the Rights Object from the SRM.
- Process the re-constructed XML **<roContainer>** as normal.

9. Replay Protection Mechanisms

9.1 General description

To prevent Rights that have been Moved from being re-installed with a replay attack or by restore from off-device storage, the DRM Agent MUST have a Move Cache with a reliable identification (GUID, see [OMADRMv2]) for all Stateless ROs that have been Moved to an SRM. The format of the GUID for Stateless ROs and BCROs are specified in section 9.1 and 9.2, respectively. The Move Cache also holds the Rights Issuer Timestamp (RITS, see section 9.1 and 9.2).

Immediately after the Rights or BCRO assets are removed during a successful Move of a Stateless RO or BCRO asset from the Device to an SRM, the DRM Agent MUST insert the <GUID, RITS> pair corresponding to the RO in the Move Cache.

If a <GUID, RITS> pair is to be inserted in the Move Cache, but the Move Cache is full and the RITS in the RO is later than the earliest RITS in the Move Cache, then the DRM Agent MUST replace the entry with the earliest RITS in the Move Cache with the new <GUID, RITS> pair. If the RITS in the RO is equal to or earlier than the earliest RITS in the Move Cache, the <GUID, RITS> pair is not inserted.

When receiving a Stateless RO or BCRO in a way other than Move (e.g. when restoring a backup from off-device storage (see [OMADRMv2.0])), the DRM Agent MUST perform the following procedure:

a) Check if the Move Cache contains the GUID of the received RO. If it does, then a) holds, otherwise a) fails.

b) If a) holds, the RO is rejected.

c) If a) fails and the Move Cache is not full, the RO is installed.

d) If a) fails, the Move Cache is full and the RITS of the RO is after the earliest RITS in the Move Cache, the RO is installed.

e) If a) fails, the Move Cache is full and the RITS of the RO is earlier than or equal to the earliest RITS in the Move Cache, the RO is rejected.

Notice that this procedure corresponds to the procedure for the replay protection mechanism for Stateless ROs uploaded to an RI from [OMADRMv2.1].

Multiple copies of the same RO (i.e. two ROs having the same GUID) SHALL NOT be installed simultaneously on the same Device. Disabled ROs (e.g. during an RO Upload or Move protocol) are considered as installed, i.e. when an RO is disabled, another copy of it MUST not be installed.

The Move Cache MUST have capacity to contain at least 100 entries. It is recommended that the Move Cache has the capacity to contain more entries.

When a Stateless RO is Moved from an SRM to the Device, and its <GUID, RITS> pair is already in the Move Cache, then this <GUID, RITS> pair MUST be removed from the Move Cache. Notice that this happens when a Stateless RO is Moved from the DRM Agent to the SRM, and later returns to the same DRM Agent via a Move operation.

9.2 Move cache for stateless XML ROs

For XML ROs, the GUID can consist of the ROID. However, a reduced size RO identification MAY be used, such as the hash over the Rights Object Container, truncated to a minimum size of 6 bytes.

The RITS consists of the <timeStamp> element in the RO.

9.3 Move cache for BCRO assets

Since a BCRO does not have a content identifier in itself, the GUID is defined as the SHA1 hash over the BroadcastRightsObjectBase() class, as defined in [DRMXBSv1.1]. This hash MAY be truncated to a minimum size of 6 bytes.

The RITS is the bcro_timestamp field in the OMADRMBroadcastRightsObjectBase() class. If this field is not present, the BCRO SHALL NOT be Moved.

Because a BCRO can have multiple assets, which can be Moved separately, it can happen that the associated <GUID, RITS> pair already exists in the Move Cache prior to adding it as a result of a Move operation. In this case, only one copy of the <GUID, RITS> pair is kept in the Move Cache.

9.4 Alternative dealing with a full Move Cache

When the Move Cache is full, the DRM Agent MAY store a part with the oldest RITS values of the Move Cache on off-device storage. In this case, the DRM Agent MUST provide for integrity protection of the externally stored part. Additionally, the DRM Agent MUST record securely that it stored a part of the Move Cache on external storage and an identification of the latest externally stored part. When an RO with a RITS before the earliest RITS in the Move Cache is delivered out-of-band or re-installed from off-device storage, the DRM Agent MUST request the externally stored part of the Move Cache, verify its integrity, verify that it is the latest externally stored part and verify if the RO has been Moved. If the integrity verification of the externally stored part fails, or if the externally stored part indicates that the RO has been Moved, the RO MUST NOT be installed. Otherwise the RO is installed. If an RO with a RITS equal to the earliest RITS in the Move Cache is delivered out-of-band or restored from a backup, both the Move Cache and the externally stored part MUST be consulted.

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
OMA-TS-SRM-V1_0-20090310-A	10 Mar 2009	Initial document to address the basic starting point Ref TP Doc# OMA-TP-2009-0099-INP_SRM_V1_0_ERP_for_Notification_and_Final_Approval
Approved Version: OMA-TS-SRM-V1_1-20110816-A	16 Aug 2011	Status changed to Approved by TP: OMA-TP-2011-0287-INP_SRM_V1_1_ERP_for_Final_Approval

Appendix B. Static Conformance Requirements (Normative)

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

Items in the SCR table are grouped by functions. Group types of SCR items represent functions as followings:

- CRT: Cryptographic algorithms
- HEL: SRM Hello message
- SAC: Mutual authentication/key exchange and Secure Authenticated Channel
- CRL: CRL processing
- OCSP: OCSP processing
- MOV: Rights Move
- LRC: Local Rights Consumption
- UTIL: Utility messages for Rights Move and Local Rights Consumption
- CERT: Functions for RI's certificates delivery and verification
- LOG: Operation log
- CAC: Move cache
- PROV: Direct Provisioning of Rights to the SRM
- BCAST: SRM Extensions for BCAST Service Support
- UPD: SRM Rights Upgrade
- S2S: S2S Rights Move

B.1 SCR for DRM Agent

The table below enumerates the client conformance requirements on DRM Agents.

Item	Function	Reference	Requirement
SRM-CRT-C-001-M	Hash Algorithms: SHA-1	Section 5.2	
SRM-CRT-C-002-M	MAC Algorithms: HMAC-SHA1	Section 5.2	
SRM-CRT-C-003-M	Symmetric Encryption Algorithms: AES-128-CBC	Section 5.2	
SRM-CRT-C-004-M	Asymmetric Encryption Algorithms: RSA-OAEP	Section 5.2	
SRM-CRT-C-005-M	Signature Algorithms: RSA-PSS	Section 5.2	
SRM-HEL-C-001-M	Device – SRM Hello	Section 6.1	
SRM-SAC-C-001-M	Mutual Authentication and Key Exchange: MAKE	Section 6.2	
SRM-SAC-C-002-M	Key Derivation Function	Section 6.3.1	
SRM-SAC-C-003-M	MAC Key update	Section 6.3.4	
SRM-SAC-C-004-O	Change SAC	Section 6.3.5	
SRM-CRL-C-001-M	CRL Number Exchange between Device and SRM	Section 6.4.1	
SRM-CRL-C-002-M	CRL Delivery from Device to SRM	Section 6.4.4	
SRM-CRL-C-003-M	CRL Delivery from SRM to Device	Section 6.4.5	

Item	Function	Reference	Requirement
SRM-CRL-C-004-M	Certificate revocation status checking using cached CRL	Section 6.4	
SRM-OCSP-C-001-O	OCSP Nonce transfer from SRM to Device	Section 6.4.2	
SRM-OCSP-C-002-O	OCSP Response transfer from Device to SRM	Section 6.4.3	
SRM-OCSP-C-003-O	OCSP Request generation	Section 6.4.2	
SRM-MOV-C-001-M	Device to SRM Move	Section 6.5	
SRM-MOV-C-002-M	Exception Recovery for Device to SRM Move	Section 6.5.1.3, 6.5.2.2, 6.5.3.3, 6.5.4.2	
SRM-MOV-C-003-M	SRM to Device Move	Section 6.6	
SRM-MOV-C-004-M	Exception Recovery for SRM to Device Move	Section 6.6.1.3, 6.6.2.2, 6.6.3.3	
SRM-MOV-C-005-M	Rights derivation from RO Payload	Section 5.1.1	
SRM-MOV-C-006-M	Move permission support	Appendix I	
SRM-LRC-C-001-M	REK transfer from SRM to Device	Section 6.7.2	
SRM-LRC-C-002-M	Exception Recovery for REK Transfer to Device	Section 6.7.2.3	
SRM-LRC-C-003-M	State Information Update	Section 6.7.3	
SRM-LRC-C-004-M	Exception Recovery for State Information Update	Section 6.7.3.3	
SRM-UTIL-C-001-M	Handle List transfer from SRM to Device	Section 6.12.1	
SRM-UTIL-C-002-M	Rights Information Transfer to Device	Section 6.12.2	
SRM-UTIL-C-003-O	Rights Information List Query	Section 6.12.3	
SRM-UTIL-C-004-M	Handle Removal from SRM	Section 6.12.4	
SRM-UTIL-C-005-M	Rights Enablement in SRM	Section 6.12.5	
SRM-UTIL-C-006-M	Rights Removal from SRM	Section 6.12.6	
SRM-UTIL-C-007-M	Dynamic Code Page Query	Section 6.12.10	
SRM-UTIL-C-008-O	Dynamic Code Page Update	Section 6.12.11	
SRM-UTIL-C-009-M	WBXML Encoding & Decoding	Section 7	
SRM-CERT-C-001-O	RI Certificate Transfer from Device to SRM	Section 6.12.7	
SRM-CERT-C-002-O	RI Certificate Transfer from SRM to Device	Section 6.12.8	
SRM-CERT-C-003-O	RI Certificate Removal from SRM	Section 6.12.9	
SRM-CERT-C-004-M	RI Certificate Chain processing and validation for Move	Section 5.1.2	
SRM-CERT-C-005-O	RI Certificate Chain processing and validation for Local Rights Consumption	Section 5.1.2	
SRM-REV-C-001-O	SRM removal detection	Section 5.5.1	
SRM-LOG-C-001-M	Operation Log	Section 5.5.2	
SRM-CAC-C-001-M	Move Cache	Section 9	
SRM-PROV-C-001-M	Generate RO Request for Direct Provisioning	Section 6.8.2	
SRM-PROV-C-002-M	Signature Query	Section 6.8.3	

Item	Function	Reference	Requirement
SRM-PROV-C-003-M	RO Acquisition for Direct Provisioning	Section 6.8.4	
SRM-PROV-C-004-M	Rights Provisioning to the SRM	Section 6.8.5, 6.8.6, 6.8.7, 6.8.8	
SRM-PROV-C-005-M	Exception Recovery for Direct Provisioning of Rights to the SRM	Section 6.8.3.3, 6.8.5.3, 6.8.7.3, 6.8.8.2	
SRM-BCAST-C-001-O	Movement of Tokens from Device to SRM	Section 6.11.1	SRM-BCAST-C-002
SRM-BCAST-C-002-O	Movement of Tokens from SRM to Device	Section 6.11.2	SRM-BCAST-C-001 SRM-BCAST-C-004 SRM-BCAST-C-005 SRM-BCAST-C-014
SRM-BCAST-C-003-O	Local Token Consumption by the Device.	Section 6.11.3	SRM-BCAST-C-004
SRM-BCAST-C-004-O	Retrieval of Token Information from the SRM	Section 6.11.4	
SRM-BCAST-C-005-O	Token Removal from the SRM	Section 6.11.5	
SRM-BCAST-C-006-O	Token Upgrade	Section 6.11.6	
SRM-BCAST-C-007-O	Movement of Broadcast Rights from Device to SRM	Section 6.11.7	SRM-BCAST-C-008
SRM-BCAST-C-008-O	Movement of Broadcast Rights from SRM to Device	Section 6.11.8	SRM-BCAST-C-007 SRM-BCAST-C-011 SRM-BCAST-C-013 SRM-BCAST-C-014
SRM-BCAST-C-009-O	Local BCAST Rights Consumption	Section 6.11.9	
SRM-BCAST-C-0010-O	BCAST Extensions to Handle List Query	Section 6.11.10.1	
SRM-BCAST-C-0011-O	Retrieval of BCRO Information from the SRM	Section 6.11.10.2	
SRM-BCAST-C-0012-O	Retrieval of BCRO Information List from the SRM	Section 6.11.10.3	
SRM-BCAST-C-0013-O	BCRO Asset Enablement on the SRM	Section 6.11.10.4	
SRM-BCAST-C-0014-O	Operation Log Extensions for BCAST	Section 5.5.3	
SRM-UPD-C-001-M	SCE 2-pass RO upgrade	Section 6.9.3	
SRM-UPD-C-002-M	Trigger for SCE 2-pass RO upgrade	Section 6.9.3	
SRM-UPD-C-003-M	Upgrade of Rights in SRM	Section 6.9.4	
SRM-S2S-C-001-M	Support S2S protocol	Section 6.10.	SRM-S2S-C-002-M; SRM-S2S-C-003-M
SRM-S2S-C-002-M	Decrypt & Encrypt the REK	Section 6.10.3	
SRM-S2S-C-003-M	Maintain two SACs with two SRMs simultaneously.	Section 6.10.	SRM-SAC-C-001-M; SRM-SAC-C-002-M; SRM-SAC-C-003-M;

Item	Function	Reference	Requirement
			SRM-SAC-C-004-O;

B.2 SCR for Server

The table below enumerates the client conformance requirements on Rights Issuers and SRM Agents.

B.2.1 SCR for RI

Item	Function	Reference	Requirement
SRM-MOV-S-001-M	Issuing Rights Object with Move permission	Appendix I	
SRM-PROV-S-001-O	ROAP Trigger for Direct Provisioning	Section 6.8.1	
SRM-PROV-S-002-O	RO Acquisition for Direct Provisioning	Section 6.8.4	
SRM-UPD-S-001-M	2-pass RO Upgrade.	Section 6.9.3	

B.2.2 SCR for SRM Agent

Item	Function	Reference	Requirement
SRM-CRT-S-001-M	Hash Algorithms: SHA-1	Section 5.2	
SRM-CRT-S-002-M	MAC Algorithms: HMAC-SHA1	Section 5.2	
SRM-CRT-S-003-M	Symmetric Encryption Algorithms: AES-128-CBC	Section 5.2	
SRM-CRT-S-004-M	Asymmetric Encryption Algorithms: RSA-OAEP	Section 5.2	
SRM-CRT-S-005-M	Signature Algorithms: RSA-PSS	Section 5.2	
SRM-HEL-S-001-M	Device – SRM Hello	Section 6.1	
SRM-SAC-S-001-M	Mutual Authentication and Key Exchange: MAKE	Section 6.2	
SRM-SAC-S-002-M	Key Derivation Function	Section 6.3.1	
SRM-SAC-S-003-M	MAC Key update	Section 6.3.4	
SRM-SAC-S-004-O	Change SAC	Section 6.3.5	
SRM-CRL-S-001-M	CRL Number Exchange between Device and SRM	Section 6.4.1	
SRM-CRL-S-002-M	CRL Store	Section 6.4.4	
SRM-CRL-S-003-M	CRL Query	Section 6.4.5	
SRM-CRL-S-004-M	Certificate revocation status checking using cached CRL	Section 6.4	
SRM-OCSP-S-001-O	Nonce generation for a secure time stamp using OCSP Response	Section 6.4.2	
SRM-OCSP-S-002-O	OCSP Response processing and validation	Section 6.4.3	
SRM-OCSP-S-003-O	Device revocation status checking using OCSP Response	Section 6.4.3	
SRM-OCSP-S-004-O	CRL issue data validation using OCSP Response	Section 6.4.4	
SRM-MOV-S-002-M	Rights Installation Setup	Section 0	
SRM-MOV-S-003-M	Rights Installation	Section 6.5.3	
SRM-MOV-S-004-M	Rights Query	Section 6.6.1	
SRM-LRC-S-001-M	REK Query	Section 6.7.2	
SRM-LRC-S-002-M	State Information Update	Section 6.7.3	

Item	Function	Reference	Requirement
SRM-UTIL-S-001-M	Handle List Generation	Section 6.12.1	
SRM-UTIL-S-002-M	Rights Information Query	Section 6.12.2	
SRM-UTIL-S-003-O	Rights Information List Query	Section 6.12.3	
SRM-UTIL-S-004-M	Handle Removal	Section 6.12.4	
SRM-UTIL-S-005-M	Rights Enablement	Section 6.12.5	
SRM-UTIL-S-006-M	Rights Removal	Section 6.12.6	
SRM-UTIL-S-007-O	Dynamic Code Page Query	Section 6.12.10	
SRM-UTIL-S-008-O	Dynamic Code Page Update	Section 6.12.11	
SRM-CERT-S-001-O	RI Certificate Store	Section 6.12.7	
SRM-CERT-S-002-O	RI Certificate Query	Section 6.12.8	
SRM-CERT-S-003-O	RI Certificate Removal	Section 6.12.9	
SRM-PROV-S-003-M	Direct Provisioning of Rights to the SRM	Section 6.8	
SRM-PROV-S-004-M	Signature Query	Section 6.8.3	
SRM-PROV-S-005-M	Provisioning Setup	Section 6.8.5	
SRM-PROV-S-006-M	Rights Provisioning	Section 6.8.7	
SRM-BCAST-S-001-O	Movement of Tokens from Device to SRM	Section 6.11.1	SRM-BCAST-S-002
SRM-BCAST-S-002-O	Movement of Tokens from SRM to Device	Section 6.11.2	SRM-BCAST-S-001 SRM-BCAST-S-004 SRM-BCAST-S-005
SRM-BCAST-S-003-O	Local Token Consumption by the Device.	Section 6.11.3	SRM-BCAST-S-004
SRM-BCAST-S-004-O	Retrieval of Token Information from the SRM	Section 6.11.4	
SRM-BCAST-S-005-O	Token Removal from the SRM	Section 6.11.5	
SRM-BCAST-S-006-O	Token Upgrade	Section 6.11.6	
SRM-BCAST-S-007-O	Movement of Broadcast Rights from Device to SRM	Section 6.11.7	SRM-BCAST-S-008
SRM-BCAST-S-008-O	Movement of Broadcast Rights from SRM to Device	Section 6.11.8	SRM-BCAST-S-007 SRM-BCAST-S-011 SRM-BCAST-S-013
SRM-BCAST-S-009-O	Local BCAST Rights Consumption	Section 6.11.9	
SRM-BCAST-S-0010-O	BCAST Extensions to Handle List Query	Section 6.11.10.1	
SRM-BCAST-S-0011-O	Retrieval of BCRO Information from the SRM	Section 6.11.10.2	
SRM-BCAST-S-0012-O	Retrieval of BCRO Information List from the SRM	Section 6.11.10.3	
SRM-BCAST-S-0013-O	BCRO Asset Enablement on the SRM	Section 6.11.10.4	
SRM-UPD-S-001-M	Reservation of storage space for the upgraded rights in SRM.	Section 6.9.2	
SRM-UPD-S-002-M	Upgrade of Rights in SRM	Section 6.9.4	
SRM-S2S-S-001-M	Support S2S protocol	Section 6.10	SRM-S2S-S-002-M
SRM-S2S-S-002-M	S2S Rights Installation Setup	Section 6.10	SRM-MOV-S-002-M;

Appendix C. Transport Mappings

This section shows SRM communication layer model and includes explanation of each layer. This section clarifies the scope of OMA SRM enabler and the work-scope of external organizations related to each type of SRM.

C.1 SRM Communication Layer Model

The SRM communication layer model divides the functions of a protocol into a series of layers. Each layer has the property that it only uses the functions of the layer below, and only exports functionality to the layer above. This section briefly dictates the specifications on how one layer interacts with another.

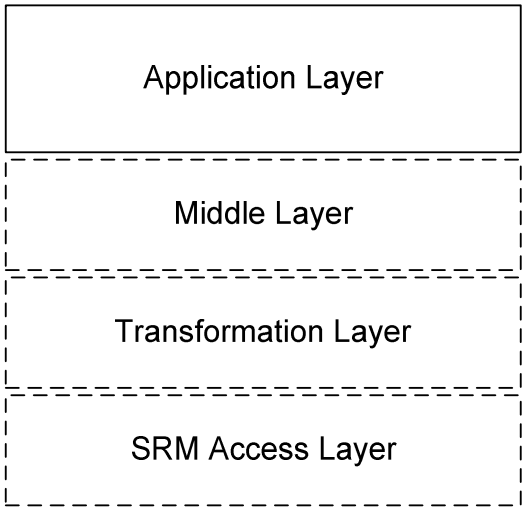

Figure 42: SRM Communication Layer

The SRM communication layer model consists of 4 layers: SRM access layer, transformation layer, middle layer and application layer. SRM access layer, transformation layer and middle layer have different property depending on each SRM type. However the application layer defines a common function of a protocol between Devices and SRM regardless of the layers below.

C.1.1 Application Layer

The application layer defines services that facilitate communication between DRM Agents and SRM Agents. This layer is independent of lower layers so that this layer is common to all SRM types.

OMA Secure Removable Media enabler specifies this layer.

C.1.2 Other Layers (Informative)

OMA Secure Removable Media enabler does not specify these layers, and these layers are defined by external organizations related to each type of SRM.

C.1.2.1 Middle Layer

The middle layer relieves the application layer of concern regarding syntactical differences in a message's data representation between Device and SRM. This layer provides functional interface defined by OMA SRM enabler for DRM agents and SRM Agents in the application layer. The implementation of this layer depends on each type of SRM.

C.1.2.2 Transformation Layer

The transformation layer defines fragmentation and de-fragmentation of the representation of digital data in Devices and SRM(s) and data blocks over a data line.

C.1.2.3 SRM Access Layer

The SRM access layer defines all the electrical and physical specifications for Device and SRM. This includes bus width, data rate, clock frequencies, and SRM form factor. The major functions and services performed by the SRM access layer are:

- Establishment and termination of a connection to a communications medium
- Modulation or conversion between data blocks and the corresponding signals transmitted over a communications channel
- Format of command line and data line
- SRM states and transition between each state

This layer also detects and corrects errors that may occur physically.

Appendix D. Method for Describing Binary Structures

D.1 Mnemonics (Data Types)

Section 2.2.6 of ISO/IEC 13818-1 [ISO/IEC13818-1] lists several data types supported by that standard. Most are not needed for SRM. The following table lists the mnemonics and data types that are needed for SRM.

Table 136: Data Types

Mnemonic	Data Type	Equivalent C Type
bslbf	Bit string, left bit first, where "left" is the order in which bit strings are written in this document. Bit strings are written as a string of 1s and 0s within single quote marks, e.g. '1000 0001'. Blanks within a bit string are for ease of reading and have no significance.	None
tcimsbf	Two's complement integer, msb (sign) bit first.	int
uimsbf	Unsigned integer, most significant bit first.	unsigned int

As seen above, the data types are all big-endian.

D.2 Comments

Comments may be interspersed in the description. Comments follow a C++ style, being preceded by two forward slashes, i.e. "//". It is suggested that they appear before the data structure or variable needing the comment. Comments are illustrated in the examples provided below.

D.3 Syntax Description

A data structure description starts with a name for the data structure. The name is begins with an upper case letter, followed by one or more upper and lower case letters (A-Z, a-z) and numbers (0-9) and finally ending with "()" (open and close parenthesis). The length of the name should be kept to a reasonable length. This document suggests that only the first letter of words be capitalized. The name of the data structure is followed by a "{" (open brace) and a newline. Next comes a list of one or more field names (one per line) and followed a "}" (close brace). The following is an example description of a data structure called *DsName()*:

```
DsName(){
   fieldName1
        .
        .
   fieldNamen
}
```

A field name represents either another data structure or a variable. If another data structure, the data structure is defined elsewhere. If a variable, then the field name is followed by two elements. Variable names follows the same rules as the name of a data structure except that it MUST begin with a lower case letter and is not followed by "()". On the same line following variable name, the next element, *nbrBits*, indicates the size of the variable in bits. The next element is the dataType of the variable, taken from Table 136 above.

The following example is for a data structure that contains an additional data structure and a 16 bit unsigned integer. The inner data structure contains a bit flag and a 32 bit signed integer.

```
Example(){
   InnerDataStructure()
   //A 16 bit unsigned integer
   uint16Var               16      uimsbf
}


InnerDataStructure(){
   //A 1 bit Boolean flag
   bitFlag                  1      bslbf
   //A 32 bit signed integer
   int32Var                32      tcimsbf
}
```

D.4 Padding

Although it is strictly not required, it is highly recommended that all integer variables and data structures start on byte boundaries. Therefore, when defining bit variables, it is up to the person defining the syntax to ensure that padding bits are defined to align the next variable or data structure on a byte boundary. The *InnerDataStructure()* example above should be rewritten as follows:

```
InnerDataStructure(){
   //A 1 bit Boolean flag
   bitFlag                  1      bslbf
   //Padding bits, reserved for future use
   rfu                      7      bslbf
   //A 32 bit signed integer
   int32Var                32      tcimsbf
}
```

D.5 Arrays

For describing an array, a C “for loop” is used. For example, the following data structure describes an array of 10 bytes:

```
FixedArrayExample(){
   for( i=0; i < 10; i++ ){
      byte                  8      uimsbf
   }
}
```

A more complex example is a variable length (0 – 255) array of signed 16 bit integers.

```
VariableArrayExample(){
   nbrOfElements            8      uimsbf
   for ( i=0; i < nbrOfElements; i++ ){
      int16                16      tcimsbf
   }
}
```

For variable sized arrays, there should be a size field (of type uimsbf) that is large enough to hold the maximum number of entries in the array. The following table lists a few of the possible ranges:

Table 137: Ranges

Number of bits	Range
8	0 - 255
16	0 - 65,535
24	0 - 16,777,215
32	0 - 4,294,967,295

D.6 Optional Variables or Data Structures

Many times there is a need for a variable or a data structure to be optional. In order to indicate whether the variable or data structure is present, a bit flag should be defined to indicate the presence. If multiple fields are optional, the indicator bit flags should be combined to minimize padding. The following example illustrates a data structure with a 16 bit integer, an optional data structure (which will not be defined), an 8 bit variable, an optional 64 bit integer and an optional fixed sized array.

```
OptionalExample(){
   int16                   16      tcimsbf
   dsPresent                1      bslbf
   int64Present             1      bslbf
   arrayPresent             1      bslbf
   //Pad to 8 bit boundary
   rfu                      5      bslbf
   if( dsPresent ){
      DataStructure()
   }
   uint8                    8      uimsbf
   if( int64Present ){
      int64                64      tcimsbf
   }
   if( arrayPresent ){
      for( i=0; i<10; i++ ){
         byte               8      uimsbf
      }
   }
}
```

For variable sized arrays, it is recommended that an optional array be indicated by the size field. So if the size field has a value of 0 (zero), then the array is not present.

Optional variables or data structures may also be indicated by the value of a previous variable as illustrated in the following example:

```
OptionalExample2(){
   status                  16      uimsbf
   if( status == 0 ){
      DataStructure()
   }
}
```

Appendix E. Data Format (Normative)

E.1 Common Data Structure

An **OctetString8** contains a variable length octet string. The minimum length is 0 octets and the maximum length is 255 octets.

```
OctetString8(){
    length                  8       uimsbf
    for( i = 0; i < length; i++ ){
        octet               8       uimsbf
    }
}
```

The fields are defined as follows:

- **length** – Length of the octet string
- **octet** – An octet (byte)

An **OctetString16** contains a variable length octet string. The minimum length is 0 octets and the maximum length is 65535 octets.

```
OctetString16(){
    length                 16       uimsbf
    for( i = 0; i < length; i++ ){
        octet               8       uimsbf
    }
}
```

The fields are defined as follows:

- **length** – Length of the octet string
- **octet** – An octet (byte)

A **RandomNumber** contains a string of random octets.

```
RandomNumber(){
    OctetString8()
}
```

A list of the algorithms is allowed by this specification.

```
Algorithm(){
    // The following algorithms are defined:
    // 0    SHA-1
    // 1    HMAC-SHA1
    // 2    AES-128-CBC
    // 3    RSA-OAEP
```

```
    // 4    DRMV2-KDF
    // 5    RSA-PSS

    algorithmId             8      uimsbf
}
```

A **Hash** contains an octet string that is the result of a cryptographic hash operation. For calculation of Asset ID hash, the SHA-1 hash algorithm is used. For all other cases, the hash algorithm used is the negotiated hash algorithm.

```
Hash(){
    // Hash Value
    OctetString8()
}
```

The **Hmac** describes an octet string that is the result of a cryptographic HMAC operation. The HMAC algorithm used is the negotiated HMAC algorithm.

```
Hmac(){
    // HMAC Value
    OctetString8()
}
```

The **EncryptedData** describes an octet string that contains the encrypted data that is a result of either a symmetric or asymmetric encryption operation. The encryption algorithm is the negotiated encryption algorithm.

```
EncryptedData(){
    // Holds the IV, length is non-zero if IV is present
    OctetString8()
    // Encrypted Data
    OctetString16()
}
```

The **Certificate** and **CertificateChain** describes an octet string comprised in certificates specified in Appendix H.

```
Certificate() {
    OctetString16()
}

CertificateChain(){
    nbrOfCerts              8      uimsbf
    for ( i = 0 ; i < nbrOfCerts ; i++ ) {
        Certificate()
    }
}
```

The **Crl** describes an octet string comprised CRL(s) specified in Appendix H.

```
Crl(){
    OctetString16()
}
```

E.2 Message Fields

E.2.1 Version

A data structure for an SRM protocol version (**`Version`**) is described as follows:

```
Version() {
    major                    4      uimsbf
    minor                    4      uimsbf
}
```

E.2.2 Status

A 16 bit value that contains the result of processing a request as follows:

```
Status() {
    status                  16      uimsbf
}
```

E.2.3 AssetID

A data structure for a DRM Asset identifier (**`AssetId`**) is described as follows. This is specified in section 5.1.5.

```
AssetId () {
    OctetString16()                 // Defined in Appendix E.1
}
```

E.2.4 Handle

A data structure for a string comprised in a Handle is described as follows:

```
Handle() {
    for ( i = 0 ; i < 10 ; i++ ) {
        byte                 8      uimsbf
    }
}
```

E.2.5 Rights

E.2.5.1 Rights Meta Data

A data structure for a Rights Meta Data (**`RightsMetaData`**) is described as follows:

```
RightsObjectVersion() {
    major                           4   uimsbf
    minor                           4   uimsbf
}


RoAlias() {
    OctetString16()                 // Defined in Appendix E.1
}


RiId() {
    OctetString8()                  // Defined in Appendix E.1
}


RiUrl() {
    OctetString16()                 // Defined in Appendix E.1
}


RiAlias() {
    OctetString16()                 // Defined in Appendix E.1
}


MetaData() {
    roAliasPresent                 1    bslbf
    riUrlPresent                   1    bslbf
    riAliasPresent                 1    bslbf
    riTimeStampPresent             1    bslbf
    extensionsContainerPresent     1    bslbf
    rfu                            3    bslbf

    RightsObjectVersion()

    if ( roAliasPresent ) {
        RoAlias()
    }

    RiId()

    if ( riUrlPresent ) {
        RiUrl()
    }
    if ( riAliasPresent) {
        RiAlias()
    }
    if ( riTimeStampPresent ) {
        riTimeStamp                 40  uimsbf
    }
    if ( extensionsContainerPresent ) {
        ExtensionsContainer()       // Defined in section 5.6.5
    }

}
```

```
RightsMetaData() {
    length                          16  uimsbf
    MetaData()
}
```

The fields are defined as follows:

- **`roAliasPresent`** - if '1', then **`RoAlias`** is present in this message
- **`riUrlPresent`** - if '1', then **`RiUrl`** is present in this message
- **`riAliasPresent`** - if '1', then **`RiAlias`** is present in this message
- **`riTimeStampPresent`** – if '1', then RiTimeStamp is present in this message
- **`extensionsContainerPresent`** – if '1', then **`ExtensionsContainer()`** is present in this data structure. In SRM v1.1, there are no extensions specified, therefore the value of the bit SHOULD be '0'. In the future, the bit MAY be set to '1'.
  Unknown extensions SHALL be ignored, but not removed, by the receiving DRM Agent or SRM Agent. If the **`MetaData()`** structure is forwarded to another entity (e.g. through Move), the **`ExtensionsContainer()`** field SHALL be included without being modified.
- **`RightsObjectVersion`** – Rights Object Version in section 5.1.1.1
- **`RoAlias`** – RO Alias in section 5.1.1.1
- **`RiId`** – RI Identifier in section 5.1.1.1
- **`RiUrl`** – RI URL in section 5.1.1.1
- **`RiAlias`** – RI Alias in section 5.1.1.1
- **`riTimeStamp`** – Rights Issuer TimeStamp in UTC (RITS, see [OMADRMv2.0]). The coding is as follows:

  RiTimeStamp = $Y << 26 + M << 22 + D << 17 + H << 12 + M << 6 + S$ where

  Y is the year minus 2000, e.g. for the year 2007 *Y*=7;
  M is the month, where *M*=0 corresponds to January;
  D is the day, where *M*=0 corresponds to the first day of the month;
  H is the hour (0-23);
  M is the minute (0-59);
  S is the second (0-59).
- **`length`** – Length of **`MetaData`**
- **`MetaData`** – Containing data of Rights Meta Data

E.2.5.2 Rights Object Container

A data structure for a Rights Object Container (**`RightsObjectContainer`**) is described as follows:

```
RightsObjectContainer() {
    //The following RO formats are defined for roFormat:
    // 0 = XML
    // 1 = WBXML
    roFormat                8      uimsbf
    OctetString16()                // Defined in Appendix E.1
}
```

The octet string comprised in the Rights Object Container is an XML document of type **oma-dd:RightsObjectContainer**. It is instantiated as a **<oma-dd:roContainer>** element and contains the **<rights>** element and the **<signature>** element from the RO payload as specified in section 5.1.1.2. The XML schema is as follows:

<!--Rights Object Container Definitions -->

<element name="roContainer" type="oma-dd:RightsObjectContainer">

<complexType name=”RightsObjectContainer”>

<sequence>

<element name=”rights” type=”o-ex:rightsType”/>

<element name=”signature” type=”ds:SignatureType”/>

</sequence>

</complexType>

E.2.5.3 State Information

A data structure for a State Information (**`StateInformation`**) is described as follows:

```
StateInformation() {
    // Length of StateInfo
    length                  16      uimsbf
    StateInfo()
}
```

The field is defined as follows:

- **`length`** – the total length of **`StateInfo()`** data (in bytes)

```
StateInfo() {
    nbrOfAssetIds           8      uimsbf
    for ( i = 0 ; i < nbrOfAssetIds ; i++ ) {        // <asset> elements
        HashOfAssetId()             // Defined in Appendix E.3
    }
    nbrOfPermissions        8      uimsbf
    for ( i = 0 ; i < nbrOfPermissions ; i++ ) {     // <permission> elements
        PermissionState()
    }
}
```

The fields are defined as follows:

- **`nbrOfAssetIds`** - the number of <asset> elements, where each <asset> element has a AssetId

- **HashOfAssetId()** - Hash of AssetId
- **nbrOfPermissions** - the number of <permission> elements, where each <permission> element refers to zero or more <asset> elements above

```
PermissionState() {
    constraintPresent               1   bslbf
    assetPresent                    1   bslbf
    playPresent                     1   bslbf
    displayPresent                  1   bslbf
    executePresent                  1   bslbf
    printPresent                    1   bslbf
    exportPresent                   1   bslbf
    movePresent                     1   bslbf
    extensionsContainerPresent      1   bslbf


    // for future extension: all zeros now
    rfu                             7   bslbf

    if ( constraintPresent ) {
        ConstraintState()
    }
    if ( assetPresent ) {
        HashOfAssetId()             // Defined in Appendix E.3
    }
    if ( playPresent ) {
        ConstraintState()
    }
    if ( displayPresent ) {
        ConstraintState()
    }
    if ( executePresent ) {
        ConstraintState()
    }
    if ( printPresent ) {
        ConstraintState()
    }
    if ( exportPresent ) {
        ConstraintState()
    }
    if ( movePresent ) {
        ConstraintState()
    }
    if( extensionsContainerPresent ) {
        ExtensionsContainer()       // Defined in section 5.6.5
    }
}
```

The fields are defined as follows:

- **constraintPresent**, **assetPresent**, **playPresent**, **displayPresent**, **executePresent**, **printPresent**, **exportPresent**, and **movePresent** - each of these flags corresponds to each of <**constraint**>, <**asset**>, <**play**>, <**display**>, <**execute**>, <**print**>, <**export**>, and <**move**> element in the <**permission**> element. If any of these elements exists, its corresponding flag is set to 1.
- **assetPresent** - if <**asset**> exists, then this permission is applied only to the AssetId whose Hash value equals to **HashOfAssetId()**

- **extensionsContainerPresent** – if '1', then **ExtensionsContainer()** is present in this data structure. In SRM v1.1, there are no extensions specified, therefore the value of the bit SHOULD be '0'. In the future, the bit MAY be set to '1'.
  Unknown extensions SHALL be ignored, but not removed, by the receiving DRM Agent or SRM Agent. If the **PermissionState()** structure is forwarded to another entity (e.g. through Move), the **ExtensionsContainer()** field SHALL be included without being modified.

```
ConstraintState() {
    countPresent                    1   bslbf
    timedCountPresent               1   bslbf
    intervalPresent                 1   bslbf
    accumulatedPresent              1   bslbf
    extensionsContainerPresent      1   bslbf
    rfu                             3   bslbf

    if ( countPresent ) {           // For count
        remainingCount              32  uimsbf
    }
    if ( timedCountPresent ) {      // For timed-count
        remainingTimedCount         32  uimsbf
    }
    if ( intervalPresent ) {
        // YYYY-MM-DDThh:mm:ssZ [ISO8601]
        // All zeros if the asset has NOT been rendered
        for ( i = 0 ; i < 20 ; i++ ) {
            byte                    8   uimsbf
        }
    }
    if ( accumulatedPresent ) {
        accumulatedTime             32  uimsbf       // upto 2^32 seconds
    }
    if( extensionsContainerPresent ) {
      ExtensionsContainer()         // Defined in section 5.6.5
    }
}
```

The fields are defined as follows:

- **countPresent**, **timedCountPresent**, **intervalPresent**, and **accumulatedPresent** - each of these flags corresponds to each of <**count**>, <**timed-count**>, <**interval**>, <**accumulated**> element in the <**constraint**> element.
- **remainingCount** - This value indicates how many times are left for the <**count**> element.
- **remainingTimedCount** - This value indicates how many times are left for the <**timed-count**> element.
- **accumulatedTime** - This value indicates how many seconds are consumed.

extensionsContainerPresent – if '1', then **ExtensionsContainer()** is present in this data structure. In SRM v1.1, there are no extensions specified, therefore the value of the bit SHOULD be '0'. In the future, the bit MAY be set to '1'. Unknown extensions SHALL be ignored, but not removed, by the receiving DRM Agent or SRM Agent. If the State Information is forwarded to another entity (e.g. through Move), the **ExtensionsContainer()** field SHALL be included without being modified.

E.2.5.4 Rights Information

A data structure for a Rights Information (**RightsInformation**) is described as follows. The Rights Meta Data, Rights Object Container, and State Information comprise a Rights Information.

```
RightsInformation() {
    stateInformationPresent         1   bslbf
    rfu                             7   bslbf
    RightsMetaData()
    RightsObjectContainer()
    if ( stateInformationPresent ) {
        StateInformation()
    }
}
```

E.2.5.5 REK

A data structure for a REK (**Rek**) is described as follows:

```
Rek() {
    for( i = 0 ; i < 16 ; i++ ){
        byte                8       uimsbf
    }

}
```

E.2.6 Rights Information List

A data structure for Rights Information List is described as follows:

```
RightsInformationList() {
    nbrOfAssetId            16      uimsbf
    for ( i = 0 ; i < nbrOfAssetId ; i++ ) {
        HashOfAssetID()             // Defined in Appendix E.3
        nbrOfRightsInfo     16      uimsbf
        for ( j = 0 ; j < nbrOfRightsInfo ; j++ ) {
            //List of Handle and Rights Information
            Handle()                // Defined in Appendix E.2.4
            RightsInformation()     // Defined in Appendix E.2.5.4
        }
    }
}
```

The fields are defined as follows:

- **nbrOfAssetId** – Number of hashes of **AssetId**s in the Rights Information List
- **HashOfAssetId** – Hash of **AssetId**. A DRM Content identified by this AssetID is associated with subsequent Rights Information.
- **nbrOfRightsInfo** – This is the number of enabled Rights in an SRM that are associated with the DRM Content identified by the **AssetId**
- **Handle** – This identifies enabled Rights in an SRM that are associated with a DRM Content identified by the **AssetId**
- **RightsInformation** – Rights Information of Rights identified by the **Handle**

E.2.7 Encrypted REK

A data structure for an encrypted REK is described as follows.

```
EncryptedRek() {
    //Contains the encrypted REK
    EncryptedData()                // Defined in Appendix E.1
}
```

E.2.8 Encrypted Handle

A data structure for an encrypted Handle is described as follows.

```
EncryptedHandle() {
    //Contains the encrypted Handle
    EncryptedData()                // Defined in Appendix E.1
}
```

E.2.9 Encrypted New Handle

A data structure for an encrypted new Handle is described as follows.

```
EncryptedNewHandle() {
    //Contains the encrypted New Handle
    EncryptedData()                // Defined in Appendix E.1
}
```

E.2.10 Tokens

E.2.10.1 Service/Program IDs

A data structure for Service/Program IDs (**`ServiceProgramIds`**) is described as follows:

```
ServiceProgramIds() {
    ServiceProgramIdsPresent        1   uimsbf
    rfu                             7   uimsbf
    if (ServiceProgramIdsPresent)
    {
        nbrOfServiceProgramIds          8           uimsbf
        for ( i=0 ; i<nbrOfServiceProgramIds ; i++ ){
            ServiceProgramFlag          1           uimsbf
            rfu                         7           uimsbf
            OctetString16()             // Defined in Appendix E.1
      }
    }
}
```

The fields are defined as follows:

- **ServiceProgramIdsPresent** – this flag indicates presence of service and/or identifiers in this data structure. If they are present, this flag is set to '1'.
- **nbrOfServiceProgramIds** – the number of service and program identifiers
- **ServiceProgramFlag** – indicates which identifier is contained
  - 0 – service_CID (see [DRMXBSv1.1] for details)
  - 1 – program_CID (see [DRMXBSv1.1] for details)

E.2.10.2 Movable

A data structure for Movable attribute (**Movable**) is described as follows:

```
Movable() {
    MovableFlag             1      uimsbf
    rfu                     7      uimsbf
}
```

The fields are defined as follows:

- **MovableFlag** – indicates if Token Move transaction is permitted or not
  - 0 – Token Move is not permitted
  - 1 – Token Move is permitted

E.2.10.3 Domain IDs

A data structure for Domain IDs (**DomainIds**) is described as follows:

```
DomainIds() {
    DomainIdsPresent                1   uimsbf
    rfu                             7   uimsbf
    if (DomainIdsPresent)
    {
        nbrOfDomainIds              8   uimsbf
        for ( i=0 ; i<nbrOfDomainIds ; i++ ){
            OctetString8()              // Defined in Appendix E.1
        }
    }
}
```

The fields are defined as follows:

- **DomainIdsPresent** – this flag indicates presence of domain identifiers in this data structure. If they are present, this flag is set to '1'.
- **nbrOfDomainIds** – the number of domain identifiers

E.2.10.4 Token Delivery ID

A data structure for Token Delivery ID (**TokenDeliveryId**) is described as follows:

```
TokenDeliveryId() {
    OctetString8()                  // Defined in Appendix E.1
}
```

E.2.10.5 Reporting Information

A data structure for Reporting Information (**ReportingInformation**) is described as follows:

```
ReportingInformation() {
    ReportingInformationPresent         1           uimsbf
    if (ReportingInformationPresent)
    {
        TokenReportingURL()
        LatestTokenConsumptionTime      40          mjdutc
        EarliestReportingTime           40          mjdutc
        LatestReportingTime             40          mjdutc
    }
}
```

The fields are defined as follows:

- **ReporingInformationPresent** – this flag indicates presence of reporting information in this data structure. If it is present, this flag is set to '1'.

```
TokenReportingURL(){
    OctetString16()                 //Defined in Appendix E.1
}
```

E.2.10.6 Token Quantity

A data structure for Token Quantity (**TokenQuantity**) is described as follows:

```
TokenQuantity() {
    TokenQuantity                   32  uimsbf
}
```

E.2.10.7 BCAST Token Container

A data structure for BCAST Token (BCAST **Token Container)** is described as follows:

```
BCASTTokenContainer() {
     RiId()                             // Defined in Appendix E.2.5
     TokenDeliveryId()                  // Defined in Appendix E.2.10.4
     ServiceProgramIds()                // Defined in Appendix E.2.10.1
     Movable()                          // Defined in Appendix E.2.10.2
     DomainIds()                        // Defined in Appendix E.2.10.3
     ReportingInformation()             // Defined in Appendix E.2.10.5
     TokenQuantity()                    // Defined in Appendix E.2.10.6
}
```

E.2.11 Broadcast Rights

E.2.11.1 BCRO Base

A data structure for BCRO Base (**BcroBase**) is described as follows:

```
BcroBase() {
    OctetString16()                // Defined in Appendix E.1
}
```

E.2.11.2 BCRO Assets

A data structure for BCRO Assets (**BcroAssets**) is described as follows:

```
BcroAssets() {
    nbrOfAssets              8      uimsbf
    for ( i=0 ; i<nbrOfAssets ; i++ ) {
        Bci()                       // Defined in Appendix E.2.11.2.2
        AssetIndex()                // Defined in Appendix E.2.11.2.1
        AssetType            1      uimsbf
        rfu                  7      uimsbf
        If ( AssetType == 0 ) {
            EncryptedSPeak()        // Defined in Appendix E.2.11.2.4
        } else {
            EncryptedCek()          // Defined in Appendix E.2.11.2.6
        }
    }
}
```

The fields are defined as follows:

- **nbrOfAssets** – number of assets (SEAKs, PEAKs and/or CEKs) in the BCRO Assets
- **AssetIndex** – Asset Index defined in section 5.1.11.2
- **AssetType** – indicates the type of the asset in BCRO Assets
  - If set to ‘0’, asset contains SEAK or PEAK
  - Otherwise, asset contains CEK
- **SPeak** – SEAK or PEAK defined in section 5.1.11.2
- **Cek** – CEK defined in section 5.1.11.2

E.2.11.2.1 Asset Index

A data structure for Asset Index (**AssetIndex**) is described as follows:

```
AssetIndex() {
        byte                8       uimsbf
}
```

E.2.11.2.2 BCI

A data structure for BCI (**Bci**) is described as follows:

```
Bci() {
    for ( i = 0 ; i < 12 ; i++ ){
         byte               8       uimsbf
    }
}
```

E.2.11.2.3 SPEAK

A data structure for SPEAK (**SPeak**) is described as follows:

```
Speak() {
    for ( i=0 ; i<32 ; i++ ) {
        byte                8       uimsbf
    }
}
```

E.2.11.2.4 Encrypted SPEAK

A data structure for Encrypted SPEAK (**EncryptedSPeak**) is described as follows:

```
EncryptedSpeak() {
    //Contains the encrypted SPEAK
    EncryptedData()                 //Defined in Appendix E.1
}
```

E.2.11.2.5 CEK

A data structure for CEK (**Cek**) is described as follows:

```
Cek() {
    for ( i=0 ; i<16 ; i++ ) {
        byte                8       uimsbf
    }
}
```

E.2.11.2.6 Encrypted CEK

A data structure for Encrypted CEK (**EncryptedCek**) is described as follows:

```
Cek() {
    //Contains the encrypted SPEAK
    EncryptedData()                //Defined in Appendix E.1
}
```

E.2.11.2.7 Asset Index List

A data structure for Asset Index List (**`AssetIndexList`**) is described as follows:

```
AssetIndexList() {
    nbrOfAssetIndexes        8      uimsbf
    for ( i=0 ; i<nbrOfAssetIndexes ; i++) {
        AssetIndex()               // Defined in Appendix E.2.11.2
    }
}
```

E.2.11.3 BCRO Information List

A data structure for BCRO Information List (**`BcroInformationList`**) is described as follows:

```
BcroInformationList() {
    nbrOfBci                 8      uimsbf
    for ( i = 0 ; i < nbrOfBcro ; i++ ) {
        Bci()                      // Defined in Appendix E.2.11.2.2
        nbrOfBcroInfo        8      uimsbf
        for ( i = 0 ; i < nbrOfBcro ; i++ ) {
            Handle()               // Defined in Appendix E.2.4
            BcroBase()             // Defined in Appendix E.2.11.1
            BcroStateInfo()        // Defined in Appendix E.2.11.3
        }
    }
}
```

The fields are defined as follows:

- **`nbrOfBci`** – Number of **`Bci`**s in the BCRO Information List
- **`nbrOfBcroInfo`** – This is the number of {BCRO Base, BCRO State Info} pairs in an SRM that are associated with the broadcast content identified by the **`Bci.`**

E.2.11.4 BCRO State Info

A data structure for BCRO State (**`BcroStateInfo`**) is described as follows:

```
BcroStateInfo() {
    length                          16  uimsbf
    nbrOfPermissions                8   uimsbf
    for ( i=0 ; i<nbrOfPermissions ; i++) {
        AssetIndexList()                // Defined in Appendix E.2.11.2.6
        rfu                         1   uimsbf
        nbrOfActions                7   uimsbf
        for ( j=0 ; j<nbrOfActions ; j++) {
                ActionType          8   uimsbf
                ConstraintsStates()     // Defined in Appendix E.2.11.4.1
        }
    }
}
```

The fields are defined as follows:

- **`length`** – length of BCROStateInfo
- **`nbrOfPermissions`** – number of permissions is this data structure; permission consists of asset indexes and associated allowed actions and constraints on these actions
- **`nbrOfAssetIndexes`** – number of asset indexes
- **`nbrOfActions`** – number of allowed actions
- **`ActionType`** – defines action type (e.g. play, move, etc.)
- **`ConstraintsStates`** – contains constraint state information per action

E.2.11.4.1 Constraints States

A data structure for Constraints States (**`ConstraintsStates`**) is described as follows:

```
ConstraintsStates() {
    ConstraintsStatesPresent        1   uimsbf
    CountConstraintPresent          1   uimsbf
    TimeCountConstraintPresent      1   uimsbf
    TimeIntervalConstraintPresent   1   uimsbf
    AccumulatedConstraintPresent    1   uimsbf
    rfu                             3   uimsbf
    if ( ConstraintsStatesPresent )
    {
        if ( CountConstraintPresent ) {
            OctetString8()              // Defined in Appendix E.1
        }
        if ( TimeCountConstraintPresent ) {
            OctetString8()              // Defined in Appendix E.1
        }
        if ( TimeIntervalConstraintPresent ) {
            StartDateTime           40  mjdutc
        }
        if (AccumulatedConstraintPresent ) {
            OctetString8()              // Defined in Appendix E.1
        }
    }
}
```

The fields are defined as follows:

- **`ConstraintsStatesPresent`** – this flag indicates when constaints states are present is this data structure
- **`CountConstraintPresent, TimeCountConstraintPresent, TimeIntervalConstraintPresent, AccumulatedConstraintPresent`** – each of these flag correspond to each of constraint descriptors defined in [DRMXBSv1.1]
- **`StartDateTime`** – contains date and time when the permission was first exercised

E.2.11.5 BCRO Signature

A data structure for BCRO Signature (**`BcroSignature`**) is described as follows:

```
BcroSignature() {
    BcroSignaturePresent            1   uimsbf
    rfu                             7   uimsbf
    if ( BcroSignaturePresent )
    {
        OctetString16()
    }
}
```

The fields are defined as follows:

- **`BcroSignaturePresent`** – this flag indicates when BCRO Signature is present is this data structure

E.3 LAID (List of Asset Identifier)

A data structure for an LAID is described as follows:

Hash of AssetID is calculated using the SHA-1 hash algorithm.

```
HashOfAssetId() {
    // Contains the hash of a AssetId
    Hash()                          // Defined in Appendix E.1
}

Laid () {
    nbrOfAssetId             8      uimsbf
    for ( i = 0 ; i < nbrOfAssetId ; i++ ) {
        HashOfAssetId()
    }
}
```

The fields are defined as follows:

- **`nbrOfAssetId`** – Number of H(**`AssetId)`** comprised in an LAID
- **`HashOfAssetId`** – H(**`AssetId`**) comprised in an LAID

E.4 Handle List

A data structure for a Handle List (**`HandleList`**) is described as follows:

```
HandleList () {
    nbrOfAssetId            16      uimsbf
    for ( i = 0 ; i < nbrOfAssetId ; i++ ) {
        HashOfAssetId()             // Defined in Appendix E.3
        Handle()                    // Defined in Appendix E.2.4
    }
}
```

The fields are defined as follows:

- **`nbrOfAssetId`** – Number of hash of **`AssetId`**s in a Handle List. If the Handle List is divided into chunks, this represents the number of hashed **`AssetId`**s in a chunk.
- **`HashOfAssetId`** – Hash of **`AssetId`**. A DRM Content identified by this AssetID is associated with a Rights in an SRM identified by a subsequent Handle.
- **`Handle`** – This identifies an enabled Rights in an SRM that is associated with the DRM Content identified by the **`AssetId`**

E.5 Dynamic Code Pages

E.5.1 Attribute Code Page

A data structure for the Dynamic Attribute Code Page (**`AttributeCodePage`**) is described as follows.

```
AttributeName() {
    OctetString8()                  // Defined in Appendix E.1
}

AttributeValue() {
    OctetString8()                  // Defined in Appendix E.1
}

CodePage() {
    rfu1                     1      bslbf
    nbrOfAttrs               7      uimsbf
    for( i = 0 ; i < nbrOfAttrs ; i++ ){
        AttributeName()
    }
    rfu2                     1      bslbf
    nbrOfAttrValues          7      uimsbf
    for( i = 0 ; i < nbrOfAttrValues ; i++ ){
        AttrributeValue()
    }
}

AttributeCodePage() {
    length                  16      uimsbf
    CodePage()
}
```

The fields are defined as follows:

- **`nbrOfAttrs`** – The number of Attribute Start Tokens in the dynamic code page. The maximum allowed value is 122.
- **`AttributeName`** – The specific string value of well-known Attribute. The array is an ordered list of all Attribute Start Tokens in the dynamic code page. The first Attribute Name in the array has the token value of 6 and each subsequent Attribute Name has a token value incremented by 1.
- **`nbrOfAttrValues`** – The number of Attribute Value Tokens in the dynamic code page. The maximum allowed value is 122.
- **`AttributeValue`** – The specific string value of a well-known Attribute Value. The array is an ordered list of all Attribute Value Tokens in the dynamic code page. The first Attribute Value in the array has the token value of 133 and each subsequent Attribute Value has a token value incremented by 1.
- **`length`** – Length of **`CodePage`**
- **`CodePage`** – Containing data of Dynamic Attribute Code Page

E.5.2 Tag Code Page

A data structure for the Dynamic Tag Code Page (**`TagCodePage`**) is described as follows.

```
TagName() {
    OctetString8()                  // Defined in Appendix E.1
}

CodePage() {
    rfu                      2      bslbf
    nbrOfTags                6      uimsbf
    for( i = 0 ; i < nbrOfTags ; i++ ){
        TagName()
    }
}

TagCodePage() {
    length                  16      uimsbf
    CodePage()
}
```

The fields are defined as follows:

- **`nbrOfTags`** – The number of Tag Names in the dynamic code page. The maximum allowed value is 59.
- **`TagName`** – The specific string value of a well-known Tag/element Name. The array is an ordered list of all Tag Names in the dynamic code page. The first Tag Name in the array has the tag identity of 6 and each subsequent Tag Name has a tag identity incremented by 1.
- **`length`** – Length of **`CodePage`**
- **`CodePage`** – Containing data of Dynamic Tag Code Page

Appendix F. SRM Transport Protocol

F.1 HTTP Mapping

An SRM MAY support an HTTP transport layer (as middle layer) to communicate with the DRM agent if it can implement a local HTTP server. In this case the DRM Agent can connect to the SRM as an HTTP client. The data are then transported and exchanged between the two entities over HTTP. This appendix defines this HTTP mapping.

The following sections describe how the data are delivered using the HTTP 1.1 protocol.

F.1.1 HTTP Headers

The HTTP Content-Type header MUST be supported. 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 request over HTTP. The Accept header specifies the media types that 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 message is received over HTTP SHOULD NOT by itself cause termination of the session.

F.1.2 SRM Requests

- The DRM Agent SHALL send SRM requests as the body of HTTP POST requests. Example:

```
POST /SRM  HTTP/1.1
Host: 127.0.0.1:3516
Content-Type: application/vnd.oma.drm.srm-pdu
... [Application Data] ...
```

In the above example the DRM Agent is using the Request-URI field for specifying the path component. The absolute URI of the SRM is specified using the HTTP Host header.

- The DRM Agent SHALL use the absolute path "/SRM" (without the quotes) to address the SRM Agent
- If the SRM has its own IP address then the DRM agent SHALL address the SRM agent via this IP address and the standard HTTP port number 80 (e.g. 192.168.0.1:80) otherwise the DRM agent SHALL use port 3516 with the local IP address 127.0.0.1 for SRM Requests (i.e. 127.0.0.1:3516)
- The DRM Agent SHOULD use persistent connections when sending requests over HTTP.
- The DRM Agent SHALL support chunk as mandated in [HTTP]
- The DRM Agent SHALL indicate to the SRM that the message is a SRM message using the HTTP Content-Type header with value application/vnd.oma.drm.srm-pdu. The following is an example of such a header field:

  Content-Type: application/vnd.oma.drm.srm-pdu
- The DRM Agent SHALL use the HTTP Accept header to indicate acceptable media types in response to SRM requests sent over HTTP. The DRM Agent MUST accept at least the following media types:
  - application/vnd.oma.drm.srm-pdu

  Example:
  - *Accept: application/vnd.oma.drm.srm-pdu*
- HTTP requests from the DRM Agent MUST contain one, and only one, SRM request message.

F.1.3 SRM Responses

- The SRM SHALL send SRM responses as the body of HTTP responses.

- The HTTP Content-Type header MUST be set to application/vnd.oma.drm.srm-pdu when a SRM message constitutes the message-body of a response. Example:

  Content-Type: application/vnd.oma.drm.srm-pdu

  If the HTTP Content-Type header value in the Response does not match the above Content-Type, the DRM Agent SHALL terminate the session.

- The SRM MUST NOT include multipart responses in an HTTP response.
- The SRM MUST include an HTTP Cache-Control header with the value no-transform when sending an integrity-protected SRM message. The no-transform directive prohibits network caches from doing any content transformations. The no-cache option must also be set in order to prevent caching of responses.

  The following is an example:

  Cache-Control: no-transform; no-cache

F.1.4 HTTP Response Codes

An SRM that refuses to perform a SRM message exchange with a DRM Agent SHOULD return a 403 (Forbidden) response. In the case of an error while processing an HTTP request, the SRM 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 SRM engine, or when the SRM engine reports an internal error (for example, the SRM schema cannot be located). If the type of a SRM request cannot be determined, the SRM MUST return a 500 (Internal Error) 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 SRM MUST respond with 200 (OK) and a suitable SRM message (possibly with SRM-related error information) in the HTTP body.

DRM Agents MUST be able to handle HTTP response codes specified here (200, 400, 403, 404, and 500).

Appendix G. SRM-API (Secure Removable Media – Application Programming Interface) (Informative)

The various SRM platforms may support different transport protocols for communication with Devices. This section defines a common set of APIs that may be used between DRM Agents and SRM Agents to support the different SRM platforms. The APIs specify the field format of message requests and responses. They are based on C language format. They are applied as follows:

- Message Request: API call with input fields
- Message Response: Result of API with output fields

G.1 Definition Structures

```
/*********************  To input Input/Output data  *********************/
typedef struct {
   unsigned long                len;          /* length of input/output */
   unsigned char*               buf;          /* buffer pointer  */
} f_bytes;

/*************************  To Error Code List ******************************/
typedef unsigned short STATUS_CODE;
```

G.2 API List

Table 138: API List

API	Function
Initialise_Message	Initialise the API processing layer.
Exchange_Message	Sends the request and receives the response.
Finalise_Message	Finalise the API processing layer.

G.2.1 Initialise_Message

Declaration :

STATUS_CODE **API_SRM_Initialise** (void *arg);

Input :

arg	Data for initialising the library. It depends on the specific SRM.

Output :

None

Return value :

SUCCESS
MEMORY_ERROR
INTERNAL_ERROR
UNKNOWN_ERROR

Function:

Initialises API processing layer.

Prior condition :

None

Post condition :

None

Notice :

The processing in this API is dependent on the type of SRM. Each SRM provides specific functions to initialise processing with this API.

G.2.2 Exchange_Message

Declaration :

STATUS_CODE **API_SRM_Exchange** (f_bytes *request, f_bytes *response);

Input :

request	Request from the DRM Agent to the SRM Agent

Output :

response	Response from the SRM Agent to the DRM Agent

Return value :

SUCCESS
BUFFER_OVERFLOW_ERROR
NO_SRM_ERROR
MEMORY_ERROR
INTERNAL_ERROR
UNKNOWN_ERROR

Function:

Sends a request to the SRM Agent and receives a response from the SRM Agent.

Prior condition :

None

Post condition :

None

Notice :

When the returned value is not SUCCESS, the value of output field *response* is not defined in the case that the API allocates the memory for processing. It means the DRM Agent does not have to deallocate the memory where *response.buf* points even if *response.buf* != NULL.

G.2.3 Finalise_Message

Declaration :

STATUS_CODE **API_SRM_Finalise** (void *arg);

Input :

arg	Data for finalizing the library. It depends on the specific SRM.

Output :

None

Return value :

SUCCESS
INTERNAL_ERROR
UNKNOWN_ERROR

Function:

Finalises the API processing layer.

Prior condition :

None

Post condition :

None

Notice :

The processing in this API is dependent on the type of SRM. Each SRM provides specific functions to finalise processing with this API.

G.3 Status Codes for API

Table 139: Status Codes

Error Code	Description
SUCCESS	Success.
BUFFER_OVERFLOW_ERROR	The length of output data exceeds the allocated buffer.
NO_SRM_ERROR	Connection to SRM is failed.
MEMORY_ERROR	Memory error in the API.
INTERNAL_ERROR	Uncategorised Internal error in the API.
UNKNOWN_ERROR	Unknown error.

Appendix H. Certificates and CRL

H.1 Certificate Profiles and Requirements

The profile for Device Certificates follows the profile of the DRM Agent Certificates in OMA DRM v2.0 [OMADRMv2.0]. The DRM Agent Certificate in OMA DRM v2.0 is referred to as Device Certificate in this specification. SRM Agents processing Device Certificates MUST meet all requirements on entities processing user certificates defined in [CertProf]. In addition, SRM Agents:

- MUST be able to process Device Certificates up to 1500 bytes long;
- MUST be able to process Device Certificates with serial numbers up to 20 bytes long;
- MUST recognize and require the presence of the oma-kp-drmAgent object identifier defined in the extKeyUsage extension in Device Certificates; and
- MUST support the cRLDistributionPoints extension

The profile for SRM Certificates follows the profile for "User Certificates for Authentication" in [CertProf] with the following modifications in Table 140:

Table 140: SRM Certificate Profile

Fields	Values
Version	Version 3 (Integer value is 2)
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 following naming attributes from [CertProf] – countryName, organizationName, organizationalUnitName, commonName, and stateOrProvinceName.
Subject Name	MUST be present and MUST use a subset of the following attributes from [CertProf] – countryName, organizationName, organizationalUnitName, commonName, and serialNumber The structure and contents of an SRM subject name shall be as follows: [countryName=<Country of manufacturer>] [organizationName=<Manufacturer company name>] [organizationalUnitName=<Manufacturing location>] [commonName=<Model name>] [serialNumber=<Unique identifier for SRM, as assigned by the Certificate Issuer>] 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 SRMs 'R Us"; CN="DRM SRM Mark V"; SN="1234567890"

Fields	Values
Extensions	The extKeyUsage extension SHALL be present, and contain (at least) the oma-kp-srmAgent key purpose object identifier: oma-kp-srmAgent OBJECT IDENTIFIER ::= {oma-kp 3} 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, and the cRLDistributionPoints extension to identify how CRL information is obtained. CAs MUST NOT include any other critical extensions.

DRM Agents processing SRM Certificates MUST meet all requirements on entities processing user certificates defined in [CertProf]. In addition, DRM Agents:

- MUST be able to process SRM Certificates up to 1500 bytes long;
- MUST be able to process SRM Certificates with serial numbers up to 20 bytes long;
- MUST recognize and require the presence of the oma-kp-srmAgent object identifier defined in the extKeyUsage extension in SRM Certificates; and
- MUST support the cRLDistributionPoints extension

The profiles for RI Certificates, CA Certificates, and OCSP Responder Certificates follow the profiles specified in Appendix D.2, D.3, and D.4 of the OMA DRM v2.0 specification [OMADRMv2.0] respectively.

H.2 CRL Profiles and Requirements

The profile for CRLs follows the CRL profile in the Certificate Revocation List (CRL) profile in [RFC3280] with the following modifications in Table 141:

Table 141: CRL Profile

Fields	Values
Version	Version 2 (Integer value is 1)
Signature	MUST be RSA with SHA-1
Issuer	MUST be present and MUST use a subset of following naming attributes from Certificate profiles in [OMADRMv2.0] – countryName, organizationName, organizationalUnitName, commonName, and stateOrProvinceName.
ThisUpdate	The issue date of this CRL
NextUpdate	The date by which the next CRL will be issued
RevokedCertificates entries	See Table 142
Extensions	CAs SHALL include the Key Identifier extension, identifying the public key corresponding to the private key used to sign a CRL. CAs SHALL include the CRL Number extension, which is used to determine when a particular CRL supersedes another CRL. CAs are recommended to not include any other extensions, but may, for compliance with [RFC3280], include the Issuing Distribution Point extension from [RFC3280] to identify how CRL information is obtained. CAs MUST NOT include any other critical extensions.

When there are no revoked Device Certificates or SRM Certificates, the revoked certificates list MUST be absent. Otherwise, revoked Device Certificates or SRM Certificates are listed by the fields in Table 142.

Table 142: RevokedCertificates Entry fields in CRL Profile

Fields	Values
UserCertificate	Revoked certificate serial number
RevocationDate	Date of revocation decision
CRL Entry Extensions	CAs may define private CRL entry extensions to carry information unique to them. Except the private CRL entry extensions, CAs MUST NOT include any other critical extensions.

Appendix I. Move Permission in Rights Object (Normative)

This document defines the extension of the OMA DRM REL specification [OMADRMv2.0] to include the Move permission in Rights Objects. The Move permission in a Rights Object grants the permission to Move the Rights Object between Devices and SRMs.

In addition, this document defines a <srmPing> constraint. For example, when this constraint is included in the <play> permission, the content can only be played if an SRM is included.

I.1 Extension of Permission Model in REL

I.1.1 Element <permission>

Element	<!ELEMENT o-ex:permission (o-ex:constraint?, o-ex:asset*, o-dd:play?, o-dd:display?, o-dd:execute?, o-dd:print?, oma-dd:export?, **o-dd:move?**)>
Semantics	In addition to the semantics as defined in OMA DRM REL [OMADRMv2.0], SRM adds an optional <move> element to the <permission> element. A single Rights Object can have only one <move> permission. When present, the parent <permission> element MUST NOT have any <asset> elements. For the other child elements (permissions), refer to the OMA DRM REL specification [OMADRMv2.0].

I.1.2 Element <move>

Element	<!ELEMENT o-dd:move (o-ex:constraint?)>
Semantics	The <move> element grants the permission to Move a Rights Object from a Device to an SRM or from an SRM to a Device. It contains an optional <constraint> element. The <move> element has the semantics of moving a Rights Object between Devices and SRMs. If the <move> element has a <constraint> child element, only the <count> or <system> constraints are allowed and all other constraints MUST NOT be present. If the <constraint> element is specified, the DRM Agent MUST grant move rights according to the <constraint> child element and the top-level <constraint> element if any. If no child <constraint> element is specified, the DRM Agent MUST grant move rights according to the top-level <constraint> element if any. If neither child nor top-level <constraint> element is specified, the DRM Agent MUST grant unlimited move rights.

I.1.3 Element <count>

Element	<!ELEMENT o-dd:count (#PCDATA)>
Semantics	In addition to the semantics as defined in OMA DRM REL [OMADRMv2.0]: If the parent <constraint> element is a child element of a <move> element, the <count> element specifies the number of times the <move> permission may be granted over the Rights Object itself.

I.1.4 Element <system>

Element	<!ELEMENT oma-dd:system (o-ex:context+)>
Semantics	In addition to the semantics as defined in OMA DRM REL [OMADRMv2.0]: The <system> constraint is allowed to also constrain the <move> permission. In the case of a <move> permission, the <system> constraint specifies the OMA DRM protocol(s) that MUST be used to move Rights Objects. In this case, the <context> elements SHALL contain a <version> element and a <uid> element. The <version> element specifies the minimum version of a protocol that MUST be used. The URN to identify the SRM protocol is registered with the OMNA. Furthermore, the <system> constraint, used with the SRM protocol identifier URN, can be used to specify that the <play> or <display> permissions MUST only be granted if the Rights Object are stored on an SRM.

I.2 Extension of Constraint Model in REL

I.2.1 Element <srmPing>

In addition to the semantics as defined in OMA DRM REL [OMADRMv2.0], SRM v1.1 adds an optional <srmPing> element to the <constraint> element.

```
<xsd:element name="srmPing" substitutionGroup="o-ex:constraintElement">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:extension base="xsd:constraintType">
        <xsd:choice>
          <xsd:element name="synchronised" type="xsd:boolean"/>
          <xsd:element name="syncThreshold" type="xsd:count/>
          <xsd:element name="checkInterval" type="xsd:duration"/>
          <xsd:any maxOccurs="unbounded" processContents="lax"/>
        </xsd:choice>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

If permission contains the <srmPing> constraint, an SRM must be present to exercise the permission. The verification of the existence of the BCAST Rights in the SRM is performed through the SRM Ping protocol (see section 6.11.9.1).

- <synchonised>: If included, this element specifies that the SRM Ping protocol MUST be synchronised with a request from BCAST Client or not. The SRM Ping protocol SHALL be executed before BCAST STKM processing as specified in [BCAST1.1-SPCP].
- <syncThreshold>: If <synchonised> is not present, this <syncThreshold> element is also not present. This optional element specifies that the number of count for STKM processing that DRM Agent does not check presence of SRM after the last presence checking. This element SHALL NOT be present if <synchronized> element is “true”.
- <checkInterval >: If included, it indicates that the SRM Ping protocol MUST be executed repeatedly and specifies the maximum amount of time between performing subsequent SRM Ping protocol. This element SHALL NOT be present if <synchronized> element is “true”.

The <srmPing> constraint SHALL include <synchonised> element or <checkInterval> element.

Appendix J. Event Counting

In order to minimize the impact of not checking the CRL validity dates, the concept of event counting with a threshold is defined in this section. Event counting is optional and consequently the normative statements in this Appendix and its subsections apply only in case event counting is implemented.

In this Appendix and its subsections, the term **entity** refers either to a DRM Agent or a SRM Agent. Each entity MUST keep an event counter, which starts at zero, and gets incremented, whenever a countable event occurs (see J.1 and J.2). When a "fresh" CRL is received, the event counter is reset; see section J.3.

The value of the predefined threshold is not defined in this specification, but set by a relevant trust model; however, the following implementation considerations may be taken into account.

- A very high threshold value effectively disables revocation status checking.
- Devices and SRMs MAY have different threshold values.
- Although this Enabler specifies a single counting mechanism, in practice, multiple counters may be used. For example, a trust model may choose to have one counter for each event type and each with its own threshold value.

The behaviour of an entity when the predefined threshold is reached is not defined in this specification but can be set by a relevant trust model. For example, a trust model may require that an entity must disallow all countable events once the threshold value is reached.

The DRM Agent MUST maintain an independent event counter for each supported trust model.

J.1 Countable DRM Agent Events

Countable DRM Agent events are:

- Moving Rights to an SRM (see section 6.5)

  Suggested counter increment operation point: Following RightsDisablementInDevice and prior to RightsInstallationRequest.

  If during recovery, HandleRemovalResponse *Status* = *Success*, then prior to enabling disabled Rights in the Device, the counter may be decremented to reverse increment operation.

- Moving Rights from an SRM (see section 6.6)

  Suggested counter increment operation point: Following RightsRemovalResponse for which *Status* = *Success*.

- Local Rights Consumption from an SRM (see section 6.7)

  Suggested counter increment operation point: Following REKQueryResponse for which *Status* = *Success*.

- Direct Provisioning of Rights to the SRM (see section 6.8)

  Suggested counter increment operation point: Following SignatureQueryResponse for which *Status* = *Success*.

- SRM Rights Upgrade (see section 6.9)

  Suggested counter increment operation point: Following RightsUpgradeResponse for which *Status* = *Success*.

- S2S Rights Move (see section 6.10)

  Suggested counter increment operation point: Following RightsRemovalResponse for which *Status* = *Success*.

- Token Move from Device to SRM (see section 6.11.1)

  Suggested counter increment operation point: Following TokenDisablementInDevice and prior to TokenInstallationRequest.

- Token Move from SRM to Device (see section 6.11.2)

Suggested counter increment operation point: Following TokenRemovalResponse for which *Status* = *Success*.

- Local Token Consumption from an SRM (see section 6.11.3)

Suggested counter increment operation point: Following TokenConsumptionResponse for which *Status* = *Success*.

- Token Upgrade (see section 6.11.6)

Suggested counter increment operation point: Following TokenUpgradeResponse for which *Status* = *Success*.

- Movement of Broadcast Rights from Device to SRM (see section 6.11.7)

Suggested counter increment operation point: Following BCRODisablementInDevice and prior to BroadcastRightsInstallationRequest.

If during recovery, HandleRemovalResponse *Status* = *Success*, then prior to enabling the disabled BCRO Asset in the Device, the counter may be decremented to reverse increment operation.

- Movement of Broadcast Rights from SRM to Device (see section 6.11.8)

Suggested counter increment operation point: Following RightsRemovalResponse for which *Status* = *Success*.

- Local BCAST Rights Consumption (see section 6.11.9)

Suggested counter increment operation point: Following REKQueryResponse for which Status = Success.

J.2 Countable SRM Agent Events

Countable SRM Agent events are:

- Moving Rights from an SRM (see section 6.6)

Suggested counter increment operation point: Following successfully handled RightsRemovalRequest and prior to associated RightsRemovalResponse.

- Local Rights Consumption from an SRM (see section 6.7)

Suggested counter increment operation point: Following successfully handled REKQueryRequest and prior to associated REKQueryResponse.

- Direct Provisioning of Rights to the SRM (see section 6.8)

Suggested counter increment operation point: Following successfully handled SignatureQueryRequest and prior to associated SignatureQueryResponse.

- SRM Rights Upgrade (see section 6.9)

Suggested counter increment operation point: Following successfully handled RightsUpgradeRequest and prior to associated RightsUpgradeResponse.

- S2S Rights Move (see section 6.10)

Suggested counter increment operation point for SRM Agent-1: Following successfully handled RightsRemovalRequest and prior to associated RightsRemovalResponse.

- Token Move from SRM to Device (see section 6.11.2)

Suggested counter increment operation point: Following successfully handled TokenRemovalRequest and prior to associated TokenRemovalResponse.

- Local Token Consumption from an SRM (see section 6.11.3)

Suggested counter increment operation point: Following successfully handled TokenConsumptionRequest and prior to associated TokenConsumptionResponse.

- Token Upgrade (see section 6.11.6)

Suggested counter increment operation point: Following successfully handled TokenUpgradeRequest and prior to associated TokenUpgradeResponse.

- Movement of Broadcast Rights from SRM to Device (see section 6.11.8)

Suggested counter increment operation point: Following successfully handled RightsRemovalRequest and prior to associated RightsRemovalResponse.

- Local BCAST Rights Consumption (see section 6.11.9)

Suggested counter increment operation point: Following successfully handled REKQueryRequest and prior to associated REKQueryResponse.

J.3 Resetting the Event Counter

Once the event counter has reached its threshold value, a “fresh” CRL for the other entity type is needed. The entity MUST have means to determine whether or not a CRL is considered "fresh" based on the entity’s “current date-time”. How the “current date-time” is determined depends on whether or not the entity supports DRM Time. Resetting the event counter to zero SHALL require a “fresh” CRL for the other entity type, i.e. a DRM Agent needs a fresh CRL for SRMs and an SRM Agent needs a fresh CRL for Devices. Note that there may be only one CRL that covers both Devices and SRMs.

If an entity supports DRM Time, then the “current date-time” is just the current DRM Time. If the cached CRL is fresh according to the current DRM Time, the entity (which supports DRM Time) can reset its event counter. Otherwise, the entity MUST get a fresh CRL before resetting its event counter. How or from where an entity gets a fresh CRL is beyond the scope of this document.

If an entity does not support DRM Time, then the entity MUST get a nonce-based secure date-time and use this as the current date-time. Once the nonce-based secure date-time is received and validated, the entity can check the freshness of its cached CRL. If the cached CRL is fresh, the entity can reset its event counter. Otherwise, the entity MUST get a fresh CRL before resetting its event counter. An entity generating a nonce MAY store the nonce in volatile or non-volatile memory. If the nonce is not available at the point of receiving the supposed nonce-based secure date-time, the date-time MUST be rejected.

Because it is anticipated that most SRMs will not support DRM Time, the SRM Agent MUST provide a nonce to the DRM Agent, which in turn MUST get a nonce-based secure date-time and provide it back to the SRM. Also, the DRM Agent MUST provide CRLs to the SRM Agent.

The SRM Agent SHOULD provide its current event counter and the threshold so that a Device can then ensure that the threshold is never reached by providing the SRM Agent with timely date-time to update the SRM.

J.4 Threshold-based Event Counting Considerations

Effective implementation of optional threshold-based event counting requires an entity to be able to periodically acquire a measure of "current date-time" that is verifiable as originating from a trusted source. If there is a malicious or unintended delay in making a response to a nonce-based date-time query available to the entity awaiting the response, then such delay increases the likelihood that the entity deems a CRL as acceptably fresh when it should not. If the date-time query occurs after the entity’s counter has already reached its threshold, then delaying the response delays the point at which the entity once again becomes useful to handle countable events.

To minimize any adversarial advantage of holding back or delaying responses to nonce-based date-time queries, a trust model may elect to prevent an entity from handling events until it receives a successful response to an outstanding date-time query or until it purges the nonce corresponding to that query, even if the entity’s counter has not reached its threshold.

In order to enable each entity to make maximally effective use of just a single event counter, a trust model may assign different weights to different countable event types. Differential weighting would give a trust model the flexibility to allow, for example, a Device which is used predominantly for Local Rights Consumption transactions to go significantly longer between CRL freshness checks than a Device that regularly engages in Move transactions to SRMs. If weights are assigned differentially, the same weight applies to the incrementing and to the decrementing (if any) when recovery is performed.

In comparison with SRM v1.0, SRM v1.1 provides a considerable number of extra countable events. This might lead to the threshold being reached earlier in time. A trust model MAY regard this and provide higher Thresholds for SRM v1.1 entities than for SRM v1.0 entities.

Appendix K. SRM and domain Rights Objects (Informative)

The SRM enabler allows RIs to issue Domain ROs with a <move> permission. This allows all Devices that are members of a Domain to receive a copy of the Domain RO and to Move their copy of the Domain RO to an SRM.

The result of Moving a Domain RO from a Device to an SRM is that the Domain RO is stored on an SRM and is disabled on the Device. In the process, the RO ceases to be a Domain RO and becomes an RO that can be consumed by any Device to which the SRM is connected to. In addition, the RO can be Moved to any other Device that is not a member of the Domain.

The Move Cache ensures that once a Device has moved its copy of the Domain RO to an SRM, it cannot reinstall the original Domain RO if this is received via a mechanism different than a Move, e.g. out-of-band delivery, restore of a backup, etc. (see section 9). This mechanism does allow the RO to be Moved back to the Device (provided the potential constraints on the <move> permission still allow for this). A DRM Agent (as specified in this enabler) will, however, not install an RO that is already installed in the Device (or accumulate or otherwise combine any available state information associated with the RO).

Note that in case all members of the Domain Move their copy of the Domain RO to an SRM (even the same SRM), then all those copies may be available for unlimited move to Devices that are not members of the Domain. An RI may choose to limit this by constraining the <move> permission. If for example the <move> permission is constrained with a <count> constraint set to the value of '1', then Domain member Devices can move their copy of the Domain RO to an SRM once. This allows Users to remove the SRM from the Domain member Device and insert it into any device and locally consume the Rights from the SRM, but they cannot move the RO from the SRM to the device.

In addition, in case a Domain RO contains stateful constraints, then the result of Moving the Domain RO to an SRM is that all state that is available to the DRM Agent is moved to the SRM. If all members of the Domain Move their copy of the stateful Domain RO to an SRM (even the same SRM), then all these stateful ROs are individually available for Moving to and/or local consumption on a single Device, which may be or not be a member of the Domain. If an RI wants to strictly limit the state that is available to a single Device to the value set by the stateful constraints in the Domain RO, then it should not include a <move> permission into a stateful Domain RO. Note that Rights derived from a stateful Domain RO cannot be moved from an SRM to a device if one instance of this Domain RO is already installed in this device.