SCE Generic Mechanisms

Approved Version 1.0 – 05 Jul 2011

Open Mobile Alliance

OMA-TS-SCE_GEN-V1_0-20110705-A

Contents

Figures

Tables

1. Scope

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

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

This specification defines the generic mechanisms that are shared by the various specification documents that constitute the SCE enabler.

2. References

2.1 Normative References

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

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

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

[OMADRM21] The OMA DRM 2.1 enabler as described in "Enabler Release Definition for DRM V2.1, Candidate Version 2.1", OMA-ERP-DRM-V2_1-20080912-C, Open Mobile Alliance™,
URL:http://www.openmobilealliance.org/

[OCSP-MP] OMA Online Certificate Status Protocol (profile of [OCSP]) V 1.0, Open Mobile Alliance™,
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

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

[RFC3447] "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", J. Jonsson, B. Kaliski, February 2003, URL:http://tools.ietf.org/html/rfc3447

[SCE-RD] "Secure Content Exchange Requirements, Draft Version 1.0",
OMA-RD-SCE-V1_0-20060908-D, Open Mobile Alliance™,
URL:http://www.openmobilealliance.org/

[SCE-AD] "Secure Content Exchange Architecture, Draft Version",
OMA-AD-SCE-V1_0-D, Open Mobile Alliance™,
URL:http://www.openmobilealliance.org/

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

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

[SCE-LRM] "DRM Local Rights Management, Draft Version",
OMA-TS-DRM-LRM- SCE-V1_0-D, Open Mobile Alliance™,
URL:http://www.openmobilealliance.org/

[SCE-DOM] "DRM User Domains, Draft Version",
OMA-TS-DRM-DOM-SCE-V1_0-D, Open Mobile Alliance™,
URL:http://www.openmobilealliance.org/

[SCE-A2A] "DRM Agent-to-Agent transfer, Draft Version",
OMA-TS-DRM-REL- SCE-V1_0-D, Open Mobile Alliance™,
URL:http://www.openmobilealliance.org/

[SCE-XSD-GEN] "Secure Content Exchange GEN XML schema", OMA-SUP-XSD_SCE_GEN-V1_0-D, Open Mobile Alliance™, URL:http://www.openmobilealliance.org/

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

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

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

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

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

2.2 Informative References

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

3. Terminology and Conventions

3.1 Conventions

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

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

3.2 Definitions

Ad Hoc Domain	A group of Devices that engage in Ad Hoc Sharing that is governed by a Domain Policy.
Ad Hoc Sharing	Sharing that is intended to allow a source Device to share specified Rights with a recipient Device in spontaneous, unplanned situations (e.g. sharing a song with a new group of friends at a party or playing a video on a hotel room TV while travelling).
Composite Object	A Media Object that contains one or more Media Objects by means of inclusion e.g. DRM messages, zip files.
Constraint	A restriction on the Permission over DRM Content.
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 smart card module (e.g. a SIM) (DRM V2.0).
Domain Authority	The entity to specify the Domain Policy for a User Domain or an Ad Hoc Domain.
Domain Enforcement Agent	The entity to enforce the Domain Policy on behalf of the Domain Authority. It may reside in the network as a service or in a User’s device.
Domain Policy	A collection of attributes which defines the policy determining characteristics of the membership of a User Domain or Ad Hoc Domain, as set by the Domain Authority that the Domain Enforcement Agent will enforce.
DRM Agent	The entity in the Device that manages Permissions for Media Objects on the Device (DRM V2.1). In this document, the DRM Agent implements some or all the functionality defined in this specification.
DRM Content	Media Objects that are consumed according to a set of Permissions in a Rights Object.
DRM Time	A secure, non user-changeable time source. The DRM Time is measured in the UTC time scale.
Local Rights Manager	An entity that is responsible for aspect(s) of Import and it may also manage an Imported-Content for a limited group of OMA DRM Agents.
Media Object	A digital work e.g. a ringing tone, a screen saver, a Java game or a Composite Object.
Requester	An entity that sends a request message to a Responder.
ReqContext	A logical context of data needed for communications with a Requester. A Responder maintains a ReqContext for each Requester it communicates with.
Responder	An entity that receives a request message from a Requester, processes the request and sends a response message to the Requester.
ResContext	A logical context of data needed for communications with a Responder. A Requester maintains a ResContext for each Responder it communicates with.

Rights	The collection of permissions and constraints defining under which circumstances access is granted to DRM Content.
Rights Issuer	An entity that issues Rights Objects to OMA DRM conformant Devices (DRM V2.1).
Rights Object	A collection of Permissions, Constraints, and other attributes which define under what circumstances access is granted to, and what usages are defined for, DRM Content. All OMA DRM Conformant Devices must adhere to the Rights Object associated with DRM Content.
User Domain	A set of v2.x and/or SCE DRM Agents that can consume User Domain Rights Objects.
User Domain Rights Object	A Rights Object that is targeted to a specific User Domain. Besides requiring membership in the User Domain, consumption may require being targeted to an SCE DRM Agent.

3.3 Abbreviations

CEK	Content Encryption Key
DA	Domain Authority
DEA	Domain Enforcement Agent
DRM	Digital Rights Management
LRM	Local Rights Manager
MAC	Message Authentication Code
MK	Message Integrity Key
OMA	Open Mobile Alliance™
OCSP	Online Certificate Status Protocol
PKI	Public Key Infrastructure
RI	Rights Issuer
RI/LRM	RI or LRM
RO	Rights Object
ROAP	Rights Object Acquisition Protocol
RSAES-OAEP	RSA Encryption Scheme-Optimal Asymmetric Encryption Padding
SA	Security Association
SCE	Secure Content Exchange
SK	Session Key
URL	Uniform Resource Locator
URI	Uniform Resource Indicator
XML	Extensible Markup Language

4. Introduction

The SCE enabler defines five entities: RI, DA, DEA, DRM Agent and LRM and various interfaces between these entities. This document defines a generic framework for the XML protocols between these entities. This framework contains:

- A client-server model for communications
- A generic set of XML Schema type definitions
- A generic set of cryptographic components
- A generic set of transport mappings for XML messages
- A generic capability signalling mechanism.

This document provides a basis for the communications specified in [SCE-DOM], [SCE-DRM], [SCE-LRM].

4.1 Communications Model

The SCE enabler assumes a client-server model for the communications between the defined entities. In general, a Requester (the client) sends a request message to a Responder (the server). The Responder processes the message and sends a response message to the Requester. However, it is possible for a Responder to send a trigger message to a Requester that causes the Requester to send a request message. The message flow is depicted in the following figure.

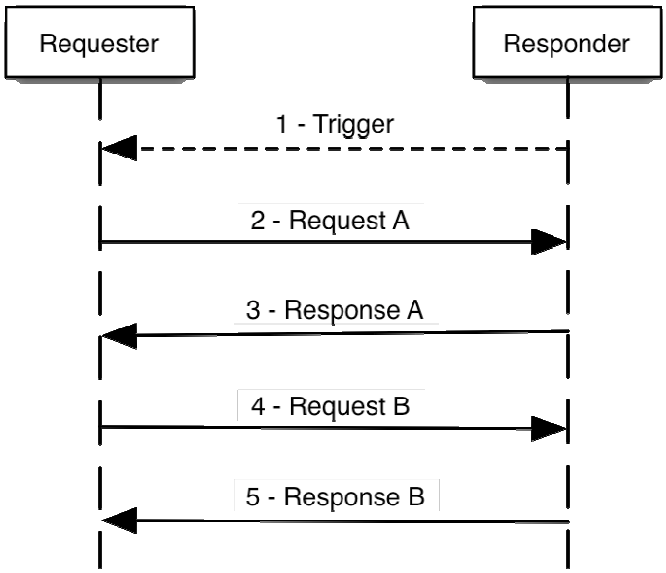

Figure 1 - Communications Model

In this flow, the Requester and Responder are SCE entities. The SCE enabler defines messages 1 through 5. By definition, messages 2 through 5 constitute the SCE protocol, which is triggered by message 1.

An SCE protocol is typically preceded by other interaction and communications (e.g. a browsing session) between both entities and may involve a human. This interaction and communications is outside the scope of the SCE enabler. In addition, the actual transport of the messages is outside the scope.

Please note that Figure 1 is a generic case. Some protocols may be initiated through other means than via a trigger. A protocol may consist of any number of requests and responses.

4.2 Contexts

In general, a Requester will "register" with a Responder to enable further communications. As a result of the registration process, the Requester will establish a logical ResContext and the Responder will establish a logical ReqContext.

4.2.1 ResContext

A Requester establishes this logical context after registering with a Responder. A Requester can have multiple ResContext's but only one per Responder. If the Requester does not have a ResContext or the ResContext is expired, then the Requester MUST only execute a registration protocol. At a minimum, the ResContext consists of the following information:

- ID of the Requester
- ID of the Responder
- The certificate chain for the Responder or a validation indicator for the chain
- The URL used to send request messages
- The selected version of the protocol
- The selected security algorithms
- The Session Key (SK)
- The Message Integrity Key (MK)

Depending on the Responder, the following information may also be present in the context.

- An alias for the Responder
- A whitelist of fully qualified domain names that are trusted
- OCSP verification data

This context expires when the Responder's certificate expires. Note that any cached OCSP responses have their own validity period, which normally will be much earlier than the expiration of the ResContext. Per [OCSP-MP], if an OCSP response does not have the nextUpdate present, then the Requester MUST not cache the OCSP response.

4.2.2 ReqContext

A Responder establishes this logical context after registering with a Requester. A Responder can have multiple ReqContext's but only one per Requester. If the Responder does not have a ReqContext or the ReqContext is expired, then the Responder MUST only accept a registration protocol from the Requester. At a minimum, the ReqContext consists of the following information:

- ID of the Requester
- ID of the Responder
- The certificate chain for the Requester or a validation indicator for the chain
- The version of the protocol
- The selected security algorithms
- The Session Key (SK)
- The Message Integrity Key (MK)

This context expires when the Requester's certificate expires.

5. Generic XML Schema

This specification defines a generic XML schema that can be used by other specifications of the SCE enabler. XML data types defined in this section are building blocks that are used by protocol messages. Specific protocol parameters are defined in the other specifications that constitute the SCE enabler. The schema for the protocol messages has been designed to be extensible.

5.1 Requirements

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

The SCE enabler does not define a collation or sorting order for attributes or element values. SCE entities MUST NOT depend on specific sorting orders for values.

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

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

In this version and minor upgrade of the specification, the namespace URI for the generic XML Schema for SCE is "*urn:oma:xml:sce:gen*". For the sake of convenience, this specification calls the generic XML Schema for SCE as the GEN schema and uses the namespace prefix for the GEN schema namespace as "gen".

5.2 Canonicalization & Digital Signatures

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

Note that all XML messages and triggers are XML 1.0 data. XML messages and triggers MUST validate against the SCE schema [SCE-XSD-GEN] and MUST not use namespace prefixes other than those used in that schema.

All canonicalization required by this specification MUST be the XML Exclusive Canonicalization without comments ([XC14N]) and MUST be signalled explicitly. The InclusiveNamespaces PrefixList of this algorithm MUST be empty. This also applies to any canonicalization step required by any of the specifications that are normatively referred to by this specification, unless such a referred specification explicitly requires a different canonicalization algorithm.

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

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

5.3 Identifiers

SCE protocols require identification of the Requester and the Responder. The only generic identifier currently defined is the hash of the public key info, as it appears in the certificate of the entity (i.e. the hash of the complete DER-encoded `subjectPublicKeyInfo` component in the certificate). The default hash algorithm is SHA-1. In case an entity holds multiple public keys, the Requester and Responder MUST select one of these public keys during registration (see Chapter 6) and MUST use the corresponding identifier in all subsequent protocols. Other identifiers are allowed but interoperability when using them is not guaranteed.

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

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

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

<complexType name="X509SPKIHash">
	<complexContent>
		<extension base="gen:KeyIdentifier">
			<sequence>
				<element name="hash" type="base64Binary"/>
			</sequence>
			<attribute name="algorithm" type="anyURI" default="http://www.w3.org/2000/09/xmldsig#sha1"/>
		</extension>
	</complexContent>
</complexType>
```

5.4 Nonce

A Nonce is defined by the following XML schema fragment:

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

The Nonce type is used to carry arbitrary values in the protocol messages. A nonce, as the name implies, must be used only once. For each message that requires a nonce element to be sent, a fresh nonce SHALL be generated randomly each time. Nonce values MUST be at least 14 octets long and are carried as a base64-encoded strings.

When a nonce value is sent in a response message (section 5.7), the value MUST be the value of the <nonce> element of the preceding request message (section 5.6).

5.5 Trigger

A Responder may send a Trigger message to cause a Requester to initiate a particular protocol. The following XML schema fragment defines a generic <drmTrigger> message:

```
<element name="drmTrigger" type="gen:DrmTrigger"/>

<complexType name="DrmTrigger">
	<sequence>
```

```
            <element name="body" type="gen:TriggerBody"/>
            <element name="signature" type="ds:SignatureType" minOccurs="0"/>
        </sequence>
        <attribute name="type" type="gen:String80" use="required"/>
        <attribute name="version" type="gen:Version"/>
</complexType>

<complexType name="TriggerBody">
        <sequence>
            <element name="resID" type="gen:Identifier"/>
            <element name="resAlias" type="gen:String80" minOccurs="0"/>
            <element name="nonce" type="gen:Nonce" minOccurs="0"/>
            <element name="reqURL" type="anyURI"/>
            <element name="trgInfo" type="gen:TriggerInformation" minOccurs="0"/>
        </sequence>
        <attribute name="id" type="ID"/>
</complexType>

<complexType name="TriggerInformation">
        <sequence>
            <any minOccurs="0" maxOccurs="unbounded" processContents="lax" namespace="##any"/>
        </sequence>
</complexType>
```

type: This attribute is used to identify the trigger type which in turn specifies the targeted protocol. The trigger types defined in the SCE Enabler are specified either in this specification or in other SCE specifications.

version: This attribute is a <major.minor> version representation of the trigger. For this version of the specification, the value of this attribute SHALL be set to "1.0".

body: This element contains child elements described below.

resID: This element identifies the Responder. The Requester MUST use this value to verify that it has a valid ResContext with the Responder. If the Requester does not have a valid ResContext with the identified Responder then the Requester MUST initiate the registration protocol of the protocol suite before initiating the protocol indicated by the *type* attribute. If the implicitly triggered registration protocol does not lead to a valid ResContext with the identified Responder, then the Requester MUST discard the trigger.

resAlias: This element, if present, contains a string value that SHALL be used by the Requester whenever it refers to the Responder in a dialog with the user and it SHALL be saved in the ResContext for future use. An example for such a dialog would be the question whether or not the user would like to register with a certain Responder after receiving a trigger. The maximum length of this element SHALL be 80 bytes.

nonce: This element provides a way to couple triggers with protocol requests. If present, the Requester will use this value as the *triggerNonce* attribute of the request that is to be triggered (see section 5.6).

reqURL: This element contains the URL that MUST be used by the Requester to send the first request message of the triggered protocol.

trgInfo: This element, if present, contains additional information that is required by the triggered protocol. This element is of type TriggerInformation which is abstract. Thus, any concreted trigger that needs this element, MUST define a concrete TriggerInformation.

signature: This element, if present, is a signature over the trigger besides the <signature> element itself. Before a Requester has registered with a Responder, the <signature> element is generally a public key signature (the default is an RSA-PSS signature). After a Requester has registered with a Responder, the <signature> element is generally a MAC signature using the negotiated algorithm (the default is an HMAC-SHA-1 signature) and the MK received during registration.

5.6 Request

A Requester sends a Request message to a Responder when executing a protocol. The following XML schema fragment defines a Request message:

```
<complexType name="Request" >
	<sequence>
		<element name="reqID" type="gen:Identifier"/>
		<element name="resID" type="gen:Identifier"/>
		<element name="nonce" type="gen:Nonce" minOccurs="0"/>
		<element name="time" type="dateTime" minOccurs="0"/>
		<element name="certificateChain" type="gen:CertificateChain" minOccurs="0"/>
		<element name="reqInfo" type="gen:RequestInformation" minOccurs="0"/>
		<element name="signature" type="base64Binary" minOccurs="0"/>
	</sequence>
	<attribute name="triggerNonce" type="gen:Nonce"/>
	<attribute name="sessionId" type="gen:String64"/>
</complexType>

<complexType name="RequestInformation">
	<sequence>
		<any minOccurs="0" maxOccurs="unbounded" processContents="lax" namespace="##any"/>
	</sequence>
	<anyAttribute namespace="##other"/>
</complexType>
```

triggerNonce: This attribute, if present, MUST contain the value of the <nonce> element sent in a <trigger> message.

sessionId: This attribute, if present, contains a protocol session identifier that allows for several, concurrent Requester-Responder sessions. The sessionId attribute is generally used for protocols that consist of more than 2 passes (e.g. registration protocol).

reqID: This element contains the identity of the Requester. If the request message is part of a registration protocol, this element contains the identity that the Requester plans to register with. Otherwise, the Requester MUST set this value to equal the Requester's ID that was stored in the ResContext during registration with the Responder.

resID: This element contains the identity of the intended Responder.

nonce: This element, if present, contains a nonce randomly generated by the Requester. Nonces are generated and used as specified in section 5.4.

time: This element, if present, contains the current DRM Time, as seen by the Requester.

certificateChain: This element, if present, contains the Requester's certificate chain. If the request is a Registration Request, this element is sent unless the preceding Hello Response contained the *Peer Key Identifier* extension and its value identified the key in the Requester's current certificate. If the request is not a Registration Request or a Hello Request, this element is sent unless it is indicated in the ResContext that the Responder has stored the necessary certificate information. When present, the value of this element SHALL be a certificate chain including the Requester's certificate. The chain MUST not include the root certificate. The Requester's certificate MUST come first in the list. Each following certificate MUST directly certify the one preceding it. If the Responder indicated trust anchor preferences in a previous Hello Response, the Requester MUST select a certificate and chain which chains back to one of the trust anchors indicated by the Responder. This mimics the features of [RFC3546]. The Responder MAY need to update this information based on the received certificate chain.

reqInfo: This element, if present, contains additional information about the request message. This element is of type RequestInformation, which is abstract. Any concrete request should define a concrete RequestInformation element.

signature: This element, if present, is a signature over the message besides the <signature> element itself. Before a Requester has registered with a Responder, the <signature> element is generally a public key signature (the default is an RSA-PSS signature). After a Requester has registered with a Responder, the <signature> element is generally a MAC signature using the negotiated algorithm (the default is an HMAC-SHA-1 signature) and the MK received during registration..

5.7 Response

A Responder sends a Response message to a Requester after processing a Request message. The following XML schema fragment defines a Response message:

```
<complexType name="Response">
	<sequence>
		<element name="reqID" type="gen:Identifier"/>
		<element name="resID" type="gen:Identifier"/>
		<element name="nonce" type="gen:Nonce" minOccurs="0"/>
		<element name="certificateChain" type="gen:CertificateChain" minOccurs="0"/>
		<element name="ocspResponse" type="base64Binary" minOccurs="0"/>
		<element name="resInfo" type="gen:ResponseInformation" minOccurs="0"/>
		<element name="signature" type="base64Binary" minOccurs="0"/>
	</sequence>
	<attribute name="status" type="gen:String80" use="required"/>
	<attribute name="errorMessage" type="gen:String1024"/>
	<attribute name="errorRedirectURL" type="anyURI"/>
	<attribute name="sessionId" type="gen:String64"/>
</complexType>

<complexType name="ResponseInformation">
	<sequence>
		<any minOccurs="0" maxOccurs="unbounded" processContents="lax" namespace="##any"/>
	</sequence>
	anyAttribute namespace="##other"/>
</complexType>
```

status: This attribute is described in section 5.7.1 below.

errorMessage: This attribute is described in section 5.7.1 below.

errorRedirectURL: This attribute is described in section 5.7.1 below.

sessionId: This attribute, if present, contains a protocol session identifier.

reqID: This element identifies the Requester. The value of this MUST be equal to the <reqID> element in the received request message. If the Responder does not have a ReqContext for the Requester and the request is considered invalid, the Responder MUST respond with a response message with the *status* attribute set to “NotRegistered”.

resID: This element identifies the Responder. The Responder MUST use same <resID> as it used during registration with this Requester.

nonce: This element contains a nonce chosen by the Responder or a nonce received from the Requester. Nonces are generated and used as specified in section 5.4.

certificateChain: This element, if present, contains the certificate chain for the Responder. If the response is a Registration Response, this element is sent unless the preceding Registration Request contained the *Peer Key Identifier* extension, the extension was not ignored by the Responder, and its value identified the key in the Responder's current certificate. If the response is not a Registration Response or a Hello Response, this element is sent unless it is indicated in the ReqContext that the Requester has stored the necessary certificate information. When present, the value of this element SHALL be a certificate chain including the Responder's certificate. The chain MUST NOT include the root certificate. The Responder’s certificate must come first in the list. Each following certificate must directly certify the one preceding it. If the Requester indicated trust anchor preferences in a previous Registration Request, the Responder SHOULD select a certificate and chain which chains back to one of the trust anchors indicated by the Requester. This mimics the features of [RFC3546]. For security reasons the Requester MUST discard the Registration Response if the hash of the complete DER-encoded *subjectPublicKeyInfo* component in the received Responder certificate does not match the value of the ResID from the preceding Hello Response message.

ocspResponse: This element, if present, contains an OCSP response for the Responder. It SHALL be a complete set of valid OCSP responses for the Responder's certificate chain. The Requester MUST NOT fail due to the presence of more than one

OCSP response element. This element will not be sent if the Requester sent the extension *No OCSP Response* in the preceding Request (and the Responder did not ignore that extension). An exception to this is when the Responder deems that the Requester's DRM Time is inaccurate. For the processing of this parameter, see further in Section 5.8.

resInfo: This element, if present, contains additional information about the Response. This element is of type ResponseInformation, which is abstract. Any concrete response should define a concrete ResponseInformation element.

signature: This element, if present, is a signature over the message besides the <signature> element itself. Before a Requester has registered with a Responder, the <signature> element is generally a public key signature (the default is an RSA-PSS signature). After a Requester has registered with a Responder, the <signature> element is generally a MAC signature using the negotiated algorithm (the default is an HMAC-SHA-1 signature) and the MK received during registration.

5.7.1 Request Processing Results

After receiving a Request message, the Responder MUST process the message and send a Response message to the Requester with the result of the processing. The result is indicated in the *status* attribute of a Response message. The following table lists the valid values for the *status* attribute.

Table 1 - Values for the *status* attribute

Value	Description
Abort	The Request message was rejected for unspecified reasons.
AccessDenied	The Requester is not authorized to contact this Responder.
DomainAccessDenied	The DRM Agent is not permitted to proxy-join or proxy-leave the User Domain.
DomainFull	The User Domain has already reached its maximum number of Devices.
InvalidCertificateChain	The Responder received or has an invalid certificate chain for the Requester.
InvalidDomain	The Responder does not recognize the requested User Domain.
InvalidRO	The RO has some invalid fields, or is unknown to the Responder.
InvalidUserDomainAuthorization	The Requester's User Domain Authorization is invalid.
LowUserDomainGeneration	The Requester's User Domain generation is lower than the Responder's User Domain generation.
MACError	The MAC value of a received message is incorrect.
MalformedRequest	The Responder failed to parse the Request message.
MovePermissionNotPresent	The Move operation requested by the Requester is not allowed because the permission is not present in the RO.
MoveServiceNotProvided	The Responder does not provide the Move via RI Service.
NoCertificateChain	The Responder needs a current certificate chain for the Requester.
NoDevPubKey	The Responder cannot return the public key that was requested by the Requester.
NotFound	Requested object was not found.
NotRegistered	The Responder does not have a ReqContext for the Requester.
NotSupported	The Request message requested a feature currently not supported by the Responder.
RequesterTimeError	The Requester’s DRM Time is deemed inaccurate by the Responder.
RightsExpired	The requested rights are no longer available (for this Requester).
SignatureError	The Responder could not verify the Requester’s signature.
Success	The Request message was processed successfully.
TriggerExpiredOrInvalid	The trigger, from which a session is initiated by the Requester, is Expired or Invalid
TrustedRootCertificateNotPresent	The Responder does not have the appropriate Trusted Root Certificate to verify the

	Requester's certificate chain.
UnknownCriticalExtension	A critical extension used by the Requester was not supported or recognised by the Responder.
UnknownRO	The Responder did not generate the RO.
UnsupportedAlgorithm	The Responder does not support a requested algorithm.
UnsupportedVersion	The Responder does not support the requested protocol version.
UpgradeServiceNotProvided	The Responder does not provide the Upgrade service to the Requester.
UserDomainAuthorizationRequired	The Responder needs a current User Domain Authorization from the Requester.
UserDomainFull	No more Devices are allowed to join the User Domain.
UserDomainNotSupported	The Requester's User Domain generation is higher than the Responder's User Domain generation, or the Responder has no valid User Domain Authorization.

If the processing of the Request message fails (i.e. the *status* attribute is not equal to "Success"), the Responder MAY add an *errorMessage* attribute containing a Responder defined description of the error. In addition, the Responder MAY add an *errorRedirectURL* attribute that points to a support web site enabling the User to recover from the error. If the *errorRedirectURL* attribute is present, then the *errorMessage* attribute MUST also be present.

Upon transmission or reception of a Response message for which Status is not "Success", the default behaviour, unless explicitly stated otherwise elsewhere, is that both the Requester and the Responder SHALL immediately close the connection and terminate the protocol. Requesters and the Responders are required to delete any session-identifiers, nonces, keys, and/or secrets associated with a failed run of the protocol.

Depending on the entity, a Requester SHOULD use the value of the *errorMessage* attribute as part of the error message presented to the User. A Requester SHOULD also either include the value of the *errorRedirectURL* attribute as part of the error message to the User, or provide the User with an option to be redirected to the *errorRedirectURL* using a browser.

5.8 Default Processing

This section specifies the default processing of Trigger, Request and Response messages. Protocol-specific processing for individual protocols is specified in relevant technical specifications.

5.8.1 Default Processing of Trigger

Upon receipt of a message of type gen:Trigger, the Requester MUST perform the following:

If the <signature> element is present, verify this signature. If the verification fails the Requester MUST discard the trigger.

For triggers other than Registration triggers, the Requester MUST use the <resID> element to verify that it has a valid ResContext with the Responder. If the Requester does not have a valid ResContext with the identified Responder then the Requester MUST initiate the registration protocol of the protocol suite before initiating the protocol indicated by the <type> attribute. If the implicitly triggered registration protocol does not lead to a valid ResContext with the identified Responder, then the Requester MUST discard the trigger.

The Requester MUST use the URL specified by the <reqURL> element in the trigger when initiating the ROAP transaction. If the trigger holds a <nonce> element, the Requester MUST use the nonce value as the *triggerNonce* attribute of the request that is to be triggered.

5.8.2 Default Processing of Request

Upon receipt of a message of type gen:Request, the Responder MUST perform the following:

For requests other than Hello requests and Registration requests, check whether the Responder has a valid ReqContent for the Requester identified by the <reqID> element. If not, the Responder MUST return a response with the *status* attribute set to "NotRegistered" and terminate the processing.

If the <triggerNonce> attribute is present, check whether the value matches the nonce in the last trigger the Responder sent to the Requester. If not, the Responder MUST discard the request and terminate the processing.

Check the <resID> element. If the value does not match any of the Responder's identities, the Responder MUST discard the request and terminate the processing.

If the <time> element is present, check whether the time is accurate. If the Responder has access to an OCSP responder, the Responder MUST use the time obtained from the OCSP responder as its reference time in order to judge the inaccuracies in the Requester's DRM Time. The Responder SHOULD allow for a reasonable drift in the Requester's DRM Time.

If the Requester's DRM time is deemed inaccurate and the request is not a Registration request, the Responder MUST return a response with the *status* attribute set to "RequesterTimeError" and terminate the processing. If the Requester's DRM time is deemed inaccurate and the request is a Registration request and the Responder has access to an OCSP responder, the Responder MUST send an OCSP request to its responder and include the nonce sent by the Requester in the OCSP request. The nonce-based OCSP response returned from the OCSP responder MUST be included in the Registration Response message sent back to the Requester.

If the <signature> element is present, verify the signature. If the verification fails the Responder MUST return a response with the *status* attribute set to an appropriate value, i.e. "SignatureError", "NoCertificateChain", "InvalidCertificateChain", or "TrustedRootCertificateNotPresent" and terminate the processing.

5.8.3 Default Processing of Response

Upon receipt of a message of type gen:Response, the Requester MUST perform the following:

Check the <reqID> element. If the value does not match any of the Requester's identities, the Requester MUST discard the response and terminate the processing.

Check the <resID> element. If the Requester does not have a pending request to the identified Responder and the protocol cannot be 1-pass, the Requester MUST discard the response and terminate the processing. If the protocol can be 1 pass but the Requester does not have a valid ResContext for the identified Responder, the Requester MUST discard the response and terminate the processing.

If the <nonce> element is present, check whether the value matches the nonce in the preceding request the Requester sent to the Responder. If not, the Requester MUST discard the response and terminate the processing.

If the <ocspResponse> element is present, check whether the nonce in the OCSP response matches the nonce sent in the preceding Registration Request. If the nonces match, the Requester MUST validate the OCSP response and the expiry time of all certificates from the OCSP responders certificate chain using the time in the *producedAt* component of the OCSP response. If the validation is successful, the Requester MUST adjust the DRM Time for the current trust model to the time in the *producedAt* component of the OCSP response. The validation of the Registration Response (and of the Responder's certificate expiry times) shall be performed afterwards by using this DRM Time.

If the <signature> element is mandatory for this particular response message, verify the signature if it is present. If the signature is not present or the verification fails, the Responder MUST discard the response and terminate the processing.

6. Generic Registration Protocol

Some SCE protocols suites specify a registration protocol. This section specifies a generic registration protocol that can be referred to in the other SCE specifications.

6.1 Registration Trigger

A Responder MAY send a Registration Trigger message to a Requester so that the Requester will initiate a 4-pass Registration Protocol. The message MUST be a <trigger> element as specified in section 5.5 and MUST be formatted as specified in the following table.

element / attribute	usage	value
id	O	Specified by the specific protocol suite
type	M	Specified by the specific protocol suite
version	M	Specified by the specific protocol suite
resID	M	As specified in section 5.5
resAlias	O	Specified by the specific protocol suite
nonce	O	As specified in section 5.5
reqURL	M	As specified in section 5.5
triggerInfo	M	Specified by the specific protocol suite

Table 2: Registration Trigger Message Parameters

The processing of the Registration Trigger is default, except for the handling of the <resID> element. The purpose of the Registration Trigger is to trigger the registration protocol, which will establish a context for the Responder in the Requester. The context for the Responder will typically not yet exist. Upon receipt of a Registration Trigger, the Requester MUST create a context for the Responder and store the <resID> and its own <reqID> with it.

6.2 Hello Request

A Requester sends a Hello Request message to a Responder as the first message of a 4-pass registration protocol of a given protocols suite. The root element of the message MUST be a <helloRequest> element of type gen:Request, as defined in the following XML schema fragment:

```
<element name=”helloRequest” type=”gen:Request”/>
```

A Hello Request MUST be formatted as specified in the table below:

element / attribute	usage	Value
triggerNonce	O	As specified in section 5.6
reqID	M	As specified in section 5.6
resID	M	As specified in section 5.6
reqInfo	M	Specified below

Table 3: Hello Request Message Parameters

The <gen:reqInfo> element under the <helloRequest> element MUST contain a <helloReqInfo> element as defined by the following XML schema fragment:

```
<element name="helloReqInfo" type="gen:HelloReqInfo"></element>
<complexType name="HelloReqInfo">
	<sequence>
		<element name="supportedAlgorithms" type="gen:SetOfAlgorithms" minOccurs="0"/>
		<element name="version" type="gen:Version"/>
		<any minOccurs="0" maxOccurs="unbounded" processContents="lax" namespace="##any"/>
	</sequence>
</complexType>

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

<complexType name="SetOfAlgorithms">
	<sequence maxOccurs="unbounded">
		<element name="selectedAlgorithm" type="anyURI"/>
	</sequence>
</complexType>
```

version: This element is a <major.minor> representation of the protocol suite version number supported by the Requester. Unless otherwise specified with the specific registration protocol, for this version of the SCE Enabler, the value of this element SHALL be “1.0”

supportedAlgorithms: This element identifies the algorithms that are supported by the Requester. Algorithms are identified using common URIs. The following algorithms and associated URIs MUST be supported by all implementations:

Hash algorithms:

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

MAC algorithms:

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

Signature algorithms:

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

Key transport algorithms:

RSAES-KEM-KDF2-KW-AES128:

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

RSAES-OAEP:

http://tools.ietf.org/html/rfc3447

Key wrapping algorithms:

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

Canonicalisation algorithms:

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

SHA-1 is defined in [SHA-1]. HMAC-SHA-1 is defined in [HMAC]. RSA-PSS-Default is RSASSA-PSS with all parameters having default values (see [PKCS-1] Appendix C). AES-WRAP is defined in [AES-WRAP]. RSA-KEM-KDF2-KW-

AES128 is defined in section 7, Key Management. Exclusive Canonicalisation is defined in [XC14N], its use is further explained in section 5.2 of this document.

Use of other algorithm URIs is optional. Since all implementations must support the algorithms above, they need not be sent. Only URIs for algorithms not in this list need to be sent in a Hello Request message.

6.3 Hello Response

A Responder send a Hello Response message to the Requester as the second message of a 4-pass registration protocol of a given protocol suite. The root element of the message MUST be a <helloResponse> element as defined in the following XML schema fragment:

```
<element name=”helloResponse” type=”gen:Response”/>
```

The response MUST be formatted per the table below:

element / attribute	usage	value
status	M	Default, as specified in section 5.7.1
sessionId	M	Specified below
errorMessage	O	Default, as specified in section 5.7
errorRedirectURL	O	Default, as specified in section 5.7
reqID	M	Default, as specified in section 5.7
resID	M	Default, as specified in section 5.7
nonce	M	Default, as specified in section 5.7
resInfo	M	Specified below

Table 4: Hello Response Message Parameters

sessionId: This attribute contains a protocol session identifier set by the Responder.

The <gen:resInfo> element under the <helloResponse> element MUST contain a <helloResInfo> element as defined by the following XML schema fragment:

```
<element name="helloResInfo" type="gen:HelloResInfo"/>

<complexType name="HelloResInfo">
      <sequence>
            <element name="selectedAlgorithms" type="SetOfAlgorithms" minOccurs="0"/>
            <element name="trustedAuthorities" type="gen:KeyIdentifiers" minOccurs="0"/>
            <element name="serverInfo" type="base64Binary" minOccurs="0"/>
            <element name="deviceDetailsRequired" type=”gen:Empty”/>
            <element name="selectedVersion" type="gen:Version"/>
            <any minOccurs="0" maxOccurs="unbounded" processContents="lax" namespace="##any"/>
      </sequence>
</complexType>

<complexType name=”Empty”/>
```

selectedAlgorithms: This element contains the cryptographic algorithms (hash algorithm, signature algorithm, MAC algorithm and key transport algorithm) to use in subsequent interactions. If the Requester indicated support of only mandatory algorithms (i.e. left out the <supportedAlgorithms> element), or the Responder only supports the mandatory algorithms, then the Responder need not send this field. Otherwise, the Responder MUST provide this parameter and MUST identify one algorithm of each type. This information is part of the ResContext and ReqContext.

trustedAuthorities: This element is a list of Requester trust anchors recognised by the Responder. This parameter is optional. The parameter is not sent if the Responder already has the Requester certificate or otherwise is able to verify a signature made by the Requester. If the parameter is present but empty, it indicates that the Requester is free to choose any Requester certificate to authenticate itself. Otherwise the Requester MUST choose a certificate chaining back to one of the recognised trust anchors. Trust anchors are identified in the same manner as Requesters and Responders.

serverInfo: This element contains server-specific information that the Requester must return unmodified, in the Registration Request. The Requester must not attempt to interpret the value of this parameter. Requesters MUST support the Server Info element being of length 512 bytes and MAY support Server Info elements of length greater than 512 bytes. Responders SHOULD keep Server Info length to 512 bytes or less.

deviceDetailsRequired: This element, if present, is used by the Responder to indicate to the Requester that the Requester needs to provide detailed information about the Device (manufacturer, model and version) in the Registration Request message that follows.

selectedVersion: This element contains the selected protocol version. The selected version will be min (Requester suggested version, highest version supported by Responder). If the registration is successful, then this information is part of the ResContext and ReqContext.

6.4 Registration Request

A Requester sends a Registration Request message to a Responder as the third message in a 4-pass registration protocol of a given protocols suite. The root element of the message MUST be a <registrationRequest> element as defined in the following XML schema fragment:

```
<element name=”registrationRequest” type=”gen:Request”/>
```

A Registration Request message MUST be formatted as specified in the table below:

element / attribute	usage	value
triggerNonce	O	Specified below
sessionId	M	Specified below
reqID	M	Default, as specified in section 5.6
resID	M	Default, as specified in section 5.6
nonce	M	Default, as specified in section 5.6
time	O	Default, as specified in section 5.6
certificateChain	O	Default, as specified in section 5.6
reqInfo	M	Specified below
signature	M	Specified below

Table 5: Registration Request Message Parameters

The <gen:reqInfo> element under the <registrationRequest> element MUST contain a <regReqInfo> element as defined in section 6.4.1.

triggerNonce: If the request message was triggered by a drmTrigger and the drmTrigger contains a <nonce> element, the "triggerNonce" attribute SHALL be included. Its value SHALL be equal to the value of the <nonce> element in the drmTrigger.

sessionId: This attribute contains the same value as included in the preceding Hello Response message.

Signature is a signature on data sent so far in the protocol. The signature is made using the Requester's private key on the two previous messages (HelloRequest, HelloResponse) and the current message (besides the *Signature* element itself). The signature method is as follows:

- The previous messages and the current one except the Signature element is canonicalized according to Section 5.2.
- The three messages are concatenated in their chronological order, starting with the HelloRequest message. The resulting data *d* is considered as input to the signature operation. The signature is calculated on *d* in accordance with the rules of the negotiated signature scheme.

The Responder MUST verify the signature on the ROAP-RegistrationRequest message.

6.4.1 RegReqInfo

The <regReqInfo> element under the <gen:reqInfo> element of a Requestration Request message is defined by the following XML schema fragment:

```
<element name="regReqInfo" type="gen:RegReqInfo"/>

<complexType name="RegReqInfo">
	<sequence>
		<element name="trustedAuthorities" type="gen:KeyIdentifiers" minOccurs="0"/>
		<element name="serverInfo" type="base64Binary" minOccurs="0"/>
		<element name="deviceDetails" type="gen:DeviceDetails" minOccurs="0"/>
		<any namespace="##any" processContents="lax" maxOccurs="unbounded"/>
	</sequence>
</complexType>

<complexType name="DeviceDetails">
	<sequence>
		<element name="manufacturer" type="roap:String64"/>
		<element name="model" type="roap:String64"/>
		<element name="version" type="roap:String64"/>
	</sequence>
</complexType>
```

trustedAuthorities: this element, if present, contains a list of Responder trust anchors recognised by the Requester. If not present, it indicates that the Responder is free to choose any certificate. Trust anchors are identified in the same way as Requesters and Responders.

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

deviceDetails: This parameter defines three fields: manufacturer, model and version. The manufacturer field identifies the Device' manufacturer, the model field identifies the Device's model and the version field identifies the Device's version as defined by its manufacturer. The <deviceDetails> element, with device details, MUST be sent by a Requester that receives a <deviceDetailsRequired> element with a “true” value in a Hello Response message.

6.5 Registration Response

A Registration Response message is sent from a Responder to a Requester as the last message in a 4-pass registration protocol of a given protocols suite. The root element of the message MUST be a <registrationResponse> element as defined in the following XML schema fragment:

```
<element name=”registrationResponse” type=”gen:Response”/>
```

The response MUST be formatted per the table below:

element / attribute	usage	value
status	M	Default, as specified in section 5.7
sessionId	M	Specified below
errorMessage	O	Default, as specified in section 5.7
errorRedirectURL	O	Default, as specified in section 5.7
reqID	M	Default, as specified in section 5.7
resID	M	Default, as specified in section 5.7
nonce	M	Default, as specified in section 5.7
certificateChain	O	Default, as specified in section 5.7
ocspResponse	O	Default, as specified in section 5.7
resInfo	O	Specified below
signature	M	Specified below

Table 6: Registration Response Message Parameters

If the processing of the Registration Request message was successful (status="Success"), then <resInfo> element under the <registrationRespone> element MUST contain an <regResInfo> element as defined in section 6.5.1.

sessionId: This attribute contains the same value as included in the preceding Registration Request message.

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

- The previous message as received (that is, including the *Signature* element) and the current one except the Signature element is canonicalized according to section 5.2.
- The two messages are concatenated in their chronological order, starting with the RegistrationRequest message. The resulting data *d* is considered as input to the signature operation. The signature is calculated on *d* in accordance with the rules of the negotiated signature scheme

The Requester MUST verify this signature. A Requester MUST NOT accept the Registration protocol as successful unless the signature verifies, the Responder certificate chain has been successfully verified, and the OCSP response, if present, indicates that the Responder certificate status is good. If the registration fails, the Requester MUST NOT store a ResContext for this Responder.

6.5.1 RegResInfo

The <regResInfo> element under the <gen:reqInfo> element of a Requestration Response message is defined by the following XML schema fragment:

```
<element name="regResInfo" type="gen:RegResInfo"/>

<complexType name="RegResInfo">
    <sequence>
        <element name="resURL" type="anyURI"/>
        <element name="encSa" type="base64Encoded" minOccurs="0"/>
        <element name="domainNameWhitelist" type="gen: DomainNameWhiteList" minOccurs="0"/>
        <any namespace="##any" processContents="lax" maxOccurs="unbounded"/>
    </sequence>
</complexType>

<complexType name="DomainNameWhiteList">
    <sequence maxOccurs="5">
        <element name="dn" type="gen:String80"/>
    </sequence>
```

```
</complexType>
```

resURL: this element contains the URL that the Requester MUST send future requests to the Responder. The value of this element MUST be a URL according to [RFC2396], and MUST be an absolute identifier.

encSa: this element, if present, contains an encrypted security association (SA). The SA contains a symmetric key (referred to as the SK) for encrypting data between the Requester and Responder, concatenated with a Message Integrity Key (MK) for providing integrity protection. For the default algorithms, the SA contains a 128-bit AES key followed by a 160-bit HMAC-SHA1 key. The SA is encrypted using the Requester's public key using RSA-OAEP [RFC3447]. This element MUST be present when an RI/RLM or Device is registering with a DEA.

domainNameWhitelist: this element, if present, contains a list of fully qualified domain names (as defined in [RFC 2396]) that are to be regarded as trusted (for example for the purposes of Silent and Preview headers). The Requester MUST store the fully qualified domain names in the ResContext for this Responder. The Requester MUST be capable of storing a minimum of 5 fully qualified domain names for each ResContext supported on the Requester.

7. Cryptographic Components

7.1 RSAES-KEM-KWS

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

KEK = KDF(I2OSP(Z,mLen) NULL, kekLen)

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

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

C_2 = WRAP(KEK, K)

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

c_1 = RSA.ENCRYPT(P,Z)

C_1 = I2OSP(c_1, mLen)

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

RSA.ENCRYPT(P,Z) = Z^e mod m

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

7.2 KDF

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

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

1) Let *T* be the empty string.

2) For *counter* from 1 to ⌈ *kLen / hLen* ⌉, do the following:

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

   Let *T* = *T* | Hash (*Z* | *D* | *otherInfo*).

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

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

7.3 AES-WRAP

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

C = AES-WRAP(KEK, K)

8. WBXML Encoding and Decoding of Trigger

This section specifies the WBXML encoding and decoding of the Trigger newly defined in SCE v1.0 enabler. Note that ROAP Triggers defined in DRM 2.1 is encoded and decoded according to [DRM-DRM-v2.1]

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.

The document type identifier for this specification is "-//OMA//SCE 1.0//EN", and is signalled as the single byte value 0x15 in the WBXML document.

8.1 Attribute Code Pages

There is one attribute code page defined. This holds attribute names and attribute values.

"-//OMA/SCE 1.0//EN" – Attribute Code Page 0				
Attribute Name	**WBXML Attribute Code**		**Attribute Value**	**WBXML Attribute Value Code**
GLOBAL TOKENS	00 – 04		GLOBAL TOKENS	80 - 84
xsi:type	05		urn:oma:drm:sce:gen	85
xmlns:sceroap	06		urn:oma:drm:sce:roap	86
xmlns:ds	07		http://www.w3.org/2000/09/xmldsig#	87
xmlns:enc	08		http://www.w3.org/2001/04/xmlenc#	88
xmlns:xsi	09		http://www.w3.org/2001/XMLSchema	89
version	0A		gen:X509SPKIHash	8A
proxy	0B		http://www.w3.org/2000/09/xmldsig#sha1	8B
id	0C		1.0	8C
type	0D		true	8D
roRequested	0E		false	8E
Algorithm	0F		moveRIRights	8F
URI	10		moveLRMRights	90
UNUSED	11 – 3F		roUpgrade	91
GLOBAL TOKENS	40 – 44		http://www.w3.org/2001/10/xml-exc-c14n#	92
UNUSED	45 – 7F		http://www.w3.org/2000/09/xmldsig#hmac-sha1	93
			#K_MAC	94
			http://www.w3.org/2001/04/xmlenc#kw-aes128	95
			UNUSED	96 – BF
			GLOBAL TOKENS	C0 – C4
			UNUSED	C5 – FF

Table x. Attribute Code Page 0

All attribute code pages 1 to 127 in the context of the public identifier "-//OMA//SCE 1.0//EN" are reserved for future use by OMA.

8.2 Tag Code Pages

There is one tag code page defined. Note that the namespace prefix sceroap is for the element defined in [SCE-DRM].

Tag Name	WBXML Attribute Code			
	No content, No attributes	Content, No attributes	No Content, Attributes	Content, Attributes
gen:drmTrigger	05	45	85	C5
body	06	46	86	C6
resID	07	47	87	C7
resAlias	08	48	88	C8
nonce	09	49	89	C9
reqURL	0A	4A	8A	CA
trgInfo	0B	4B	8B	CB
keyIdentifier	0C	4C	8C	CC
hash	0D	4D	8D	CD
algorithm	0E	4E	8E	CE
sceroap:moveRIRightsTriggerInfo	0F	4F	8F	CF
signature	10	50	90	D0
ds:SignedInfo	11	51	91	D1
ds:SignatureValue	12	52	92	D2
ds:KeyInfo	13	53	93	D3
ds:CanonicalizationMethod	14	54	94	D4
ds:SignatureMethod	15	55	95	D5
ds:Reference	16	56	96	D6
ds:Transforms	17	57	97	D7
ds:DigestMethod	18	58	98	D8
ds:DigestValue	19	59	99	D9
ds:RetrievalMethod	1A	5A	9A	DA
encKey	1B	5B	9B	DB
xenc:EncryptionMethod	1C	5C	9C	DC
xenc:CipherData	1D	5D	9D	DD
xenc:CipherValue	1E	5E	9E	DE
sceroap:moveLRMRightsTriggerInfo	1F	5F	9F	DF
sceroap:roUpgradeTriggerInfo	20	60	A0	E0
recipientInfo	21	61	A1	E1
roID	22	62	A2	E2
roAlias	23	63	A3	E3
upgradeInfo	24	64	A4	E4
Reserved for Future Use	25-3F	65-7F	A5-BF	E5-FF

Table x. Tag Code Page 0

All tag code pages 1 to 127 in the context of the public identifier "-//OMA//SCE 1.0//EN" are reserved for future use by OMA.

8.3 Processing Rules

8.3.1 MIME Type

Trigger messages encoded in WBXML format are identified with the following mime type:

application/vnd.oma.sce.trigger+wbxml (SCE Trigger)

8.3.2 Binary Data Representation

Binary data is not used in Exclusive Canonical Triggers. However Responder MAY use opaque data to represent whitespace in the canonical XML.

8.3.3 base64Binary Representation

Some elements in Exclusive Canonical SCE Triggers hold base64Binary data. Such data is encoded in the WBXML form directly as a literal string. This does not achieve maximum efficiency but intended to simplify the processing of the WBXML decoder.

8.3.4 Responder

Responders MAY support WBXML encoding of Triggers.

Responders are recommended not to embed whitespace between elements in Triggers that are to be WBXML encoded.

8.3.5 Requester

Requester MUST support WBXML decoding of Triggers.

If a Requester supports WBXML decoding of SCE Trigger, it must signal this in the HTTP Accept header of any of its requests by including the applicable mime type (application/vnd.oma.sce.trigger+wbxml).

8.3.6 Normal Processing and Transcoding

After receiving a message in WBXML format, and if that message in WBXML format is supported by the Requester, that Requester MUST decode the message into Exclusive Canonical XML format before any other processing is applied.

9. Security Considerations (Informative)

9.1 Security scope for SCE

The scope of SCE is to enable the distribution and consumption of digital content in a controlled manner. SCE work addresses various technical aspects of this system by providing appropriate specifications for content formats, protocols, and a rights expression language.

The SCE trust model is built on a PKI. The entities defined in this enabler trust each other to behave correctly if their certificates are verifiable and not revoked. The SCE specified content formats, protocols and other data structures are designed to provide adequate security of Protected Content, provided that the entities involved in the protocols behave as specified in this enabler. It is NOT in scope for SCE to specify technical or other mechanisms that in some way ensure that implementations of SCE cannot be altered such that the security of the Protected Content is compromised. It is on the other hand IN scope for the SCE to define mechanisms that enable the trust authority to deal with security compromises, should they occur and are detected.

It is anticipated that trust authorities enabling the practical use of SCE will require implementations of SCE to ALSO implement additional mechanisms to safeguard against compromises of the implementation itself. Figure 2 depicts an example implementation of SCE.

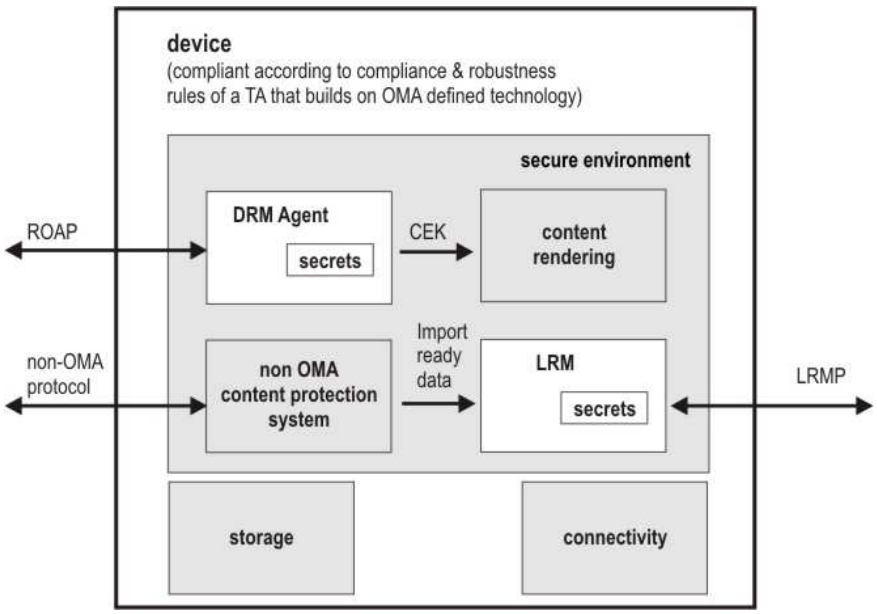

Figure 2: Example implementation

The device in Figure 2 is compliant to the compliance and robustness rules of a certain trust authority. In return the trust authority has provided the manufacturer of this device with a (set of) valid certificates and associated private keys for the functional entities that are implemented in the device. In addition to the SCE defined entities, DRM Agent and LRM, four functional entities not defined by OMA and a "Secure Environment".

To establish trust with SCE defined entities deployed in other devices or servers (e.g. when using ROAP to request a Rights Object), the SCE defined entities in Figure 2 will use the certificates issued by the trust authority and the associated private key. To assure confidentiality of these private keys and in general correct behaviour of the SCE entities, the trust authority is likely to require these entities to be implemented in some "secure environment" to protect against compromises of the device. The specification of such a "secure environment" is OUT of scope for SCE. Please note that it is IN scope for SCE to ensure that the protocols between SCE entities are secure when executed over any type of connectivity.

Since the actual decryption and rendering of Protected Content is not performed by the DRM Agent, the DRM Agent will likely transfer the CEK for a Protected Content to some sort of content rendering entity, after enforcing the permissions and constraints. This process and the content rendering entity itself are OUT of scope for SCE. But since the process involves the CEK, the trust authority is likely to require this to take place in the "secure environment". Please note that it is IN scope for SCE to ensure that the Protected Content is wrapped in a file that can be stored on any storage medium and transferred via any connectivity.

The device in Figure 2 also implements an LRM, which takes Import Ready data from a non-OMA content protection system. This process is also OUT of scope for SCE but since also in this process the transfer of secrets is required, it is likely to be implemented in the "secure environment". In this case the "secure environment" has to meet the requirements of not only the trust authority for the OMA system but also the requirements of the equivalent body for the non-OMA content protection system. All of this is OUT of scope for SCE.

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
OMA-TS-SCE_GEN-V1_0-20110705-A	05 Jul 2011	Status changed to Approved by TP: OMA-TP-2011-0233-INP_SCE_V1_0_ERP_for_Final_Approval

Appendix B. Static Conformance Requirements (Normative)

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

B.1 SCR for DRM Agent

Item	Function	Reference	Requirement
SCE-GEN-DRMAGENT-C-001-O	Generic XML Schema	5	SCE-GEN-DRMAGENT-C-003-O SCE-GEN-DRMAGENT-C-004-O SCE-GEN-DRMAGENT-C-005-O SCE-GEN-DRMAGENT-C-006-O SCE-GEN-DRMAGENT-C-007-O SCE-GEN-DRMAGENT-C-008-O
SCE-GEN-DRMAGENT-C-002-M	Canonicalization and digital signatures	5.2	
SCE-GEN-DRMAGENT-C-003-O	Identifiers	5.3	
SCE-GEN-DRMAGENT-C-004-O	Nonce	5.4	
SCE-GEN-DRMAGENT-C-005-O	Generic Trigger	5.5	
SCE-GEN-DRMAGENT-C-006-O	Generic Request	5.6	
SCE-GEN-DRMAGENT-C-007-O	Generic Response	5.7	
SCE-GEN-DRMAGENT-C-008-O	Generic registration protocol	6	
SCE-GEN-DRMAGENT-C-009-M	RSAES-KEM-KWS	7.1	
SCE-GEN-DRMAGENT-C-010-M	KDF	7.2	
SCE-GEN-DRMAGENT-C-011-M	AES-WRAP	7.3	

B.2 SCR for RI and LRM without the oma-kp-localRightsManagerDomain key purpose

The SCR tables for an RI and an LRM without the oma-kp-localRightsManagerDomain key purpose only are equal to the SCR table for the DRM Agent as specified in B.1.

B.3 SCR for LRM with at least the oma-kp-localRightsManagerDomain key purpose and the DEA

The following CR table holds for LRMs with at least the oma-kp-localRightsManagerDomain key purpose, or the DEA:

Item	Function	Reference	Requirement
SCE-GEN-LRMDOM-C-001-M	Generic XML Schema	5	
SCE-GEN-LRMDOM-C-002-M	Canonicalization and digital signatures	5.2	
SCE-GEN-LRMDOM-S-003-M	Identifiers	5.3	
SCE-GEN-LRMDOM-S-004-M	Nonce	5.4	
SCE-GEN-LRMDOM-S-005-M	Generic Trigger	5.5	
SCE-GEN-LRMDOM-S-006-M	Generic Request	5.6	
SCE-GEN-LRMDOM-S-007-M	Generic Response	5.7	
SCE-GEN-LRMDOM-S-008-M	Generic registration protocol	6	
SCE-GEN-LRMDOM-S-009-M	RSAES-KEM-KWS	7.1	
SCE-GEN-LRMDOM-S-010-M	KDF	7.2	
SCE-GEN-LRMDOM-S-011-M	AES-WRAP	7.3	