[MS-RMPRS]:
Rights Management Services (RMS):
Server-to-Server Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
06/08/2007	1.0	Major	Initial Availability
07/10/2007	1.0.1	Editorial	Revised and edited the technical content.
08/17/2007	1.0.2	Editorial	Revised and edited the technical content.
09/21/2007	1.0.3	Editorial	Revised and edited the technical content.

Date	Revision History	Revision Class	Comments
10/26/2007	1.0.4	Editorial	Revised and edited the technical content.
01/25/2008	1.0.5	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

This document specifies the Rights Management Services (RMS): Server-to-Server Protocol. The RMS: Server-to-Server Protocol is used to communicate information between RMS servers and consists of five separate interfaces:

- **Service Location**: Using the **Service Location** interface, one RMS server provides another with URLs for services that are requested.
- **Sub-Enrollment**: Using the **Sub-Enrollment** interface, one RMS server bootstraps itself as a subordinate of another RMS server. In this process, the main server grants the subordinate server the right to perform only licensing tasks by issuing it a Server Licensor Certificate (SLC) that indicates this right.
- **Get Licensor Certificate**: One RMS server acquires the public server licensor certificate (SLC) of another using the **Get Licensor Certificate** interface in order to establish trust. This trust allows the requesting RMS server to trust RMS account certificates (RACs) issued by the responding RMS server.
- **Group Expansion over SOAP**: An RMS server uses the **Group Expansion over SOAP** interface to ask another RMS server whether or not a specific user is a member of a specific group.
- **Group Expansion over Remoting**: This interface provides the same service as the **Group Expansion over Remoting** interface, using a .NET Framework Remoting-based protocol.

The **Group Expansion over Remoting** interface uses a binary-formatted .NET Framework Remoting-based protocol over HTTP. The other interfaces (**Service Location**, **Sub-Enrollment**, **Get Licensor Certificate**, and **Group Expansion over SOAP**) all use a SOAP-based protocol over HTTP.<1>

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Active Directory (AD)
Certificate Chain
Directory
Domain
Endpoint
Forest
Globally Unique Identifier (GUID)
Interface Definition Language (IDL)
Remote Procedure Call (RPC)
Secure Sockets Layer (SSL)
Security Identifier (SID)
Well-Known Endpoint

The following terms are defined in [MS-RMPR]:

Certificate
Consumer
License
License Chain
Protected Content
Publishing License (PL)
RMS Account Certificate (RAC)

Server Licensor Certificate (SLC)
Use License (UL)

The following terms are specific to this document:

Hash Table: A data structure that associates data keys with values to improve lookup efficiency.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, http://www.opengroup.org/public/pubs/catalog/c706.htm

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MS-RMPR] Microsoft Corporation, "Rights Management Services (RMS): Client-to-Server Protocol Specification", March 2007.

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions", July 2006.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

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

[RFC2616] Fielding, R., et al., "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, http://www.ietf.org/rfc/rfc2616.txt

[SOAP1.1] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D., "Simple Object Access Protocol (SOAP) 1.1", May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[SOAP1.2/1] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., and Nielsen, H.F., "SOAP Version 1.2 Part 1: Messaging Framework", W3C Recommendation, June 2003, http://www.w3.org/TR/2003/REC-soap12-part1-20030624

[SOAP1.2/2] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., and Nielsen, H.F., "SOAP Version 1.2 Part 2: Adjuncts", W3C Recommendation, June 2003, http://www.w3.org/TR/2003/REC-soap12-part2-20030624

[WSDL] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S., "Web Services Description Language (WSDL) 1.1", W3C Note, March 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[WSDLExt] Nielsen, H.F., Christensen, E., and Farrell, J., "WS-Attachments", June 2002, http://xml.coverpages.org/draft-nielsen-dime-soap-01.txt

If you have any trouble finding [WSDLExt], please check here.

[XML1.0] Bray, T., Paoli, J., Sperberg-McQueen, C.M., and Maler, E., "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation, October 2000, http://www.w3.org/TR/2000/REC-xml-20001006

[XMLNS] World Wide Web Consortium, "Namespaces in XML 1.0 (Second Edition)", August 2006, http://www.w3.org/TR/REC-xml-names/

[XMLSCHEMA1] Thompson, H.S., Ed., Beech, D., Ed., Maloney, M., Ed., and Mendelsohn, N., Ed., "XML Schema Part 1: Structures", W3C Recommendation, May 2001, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

[XMLSCHEMA2] Biron, P.V., Ed. and Malhotra, A., Ed., "XML Schema Part 2: Datatypes", W3C Recommendation, May 2001, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

[XPATH] Clark, J. and DeRose, S., "XML Path Language (XPath), Version 1.0", W3C Recommendation, November 1999, http://www.w3.org/TR/xpath

[XRML] ContentGuard, Inc., "XrML... eXtensible rights Markup Language", 2005, http://www.xrml.org/XrML_12.asp

If you have any trouble finding [XRML], please check here.

1.2.2 Informative References

[KERBKEY] Microsoft Corporation, "KERB_CRYPTO_KEY", http://msdn2.microsoft.com/en-us/library/aa378058.aspx

[MS-ADTS] Microsoft Corporation, "Active Directory Technical Specification", July 2006.

1.3 Protocol Overview (Synopsis)

Rights Management Services (RMS) is a client/server technology that provides support for information protection through content encryption and fine-grained policy definition and enforcement. The Rights Management Services (RMS): Client-to-Server Protocol, as specified in [MS-RMPR], enables end users to create and consume **protected content**.

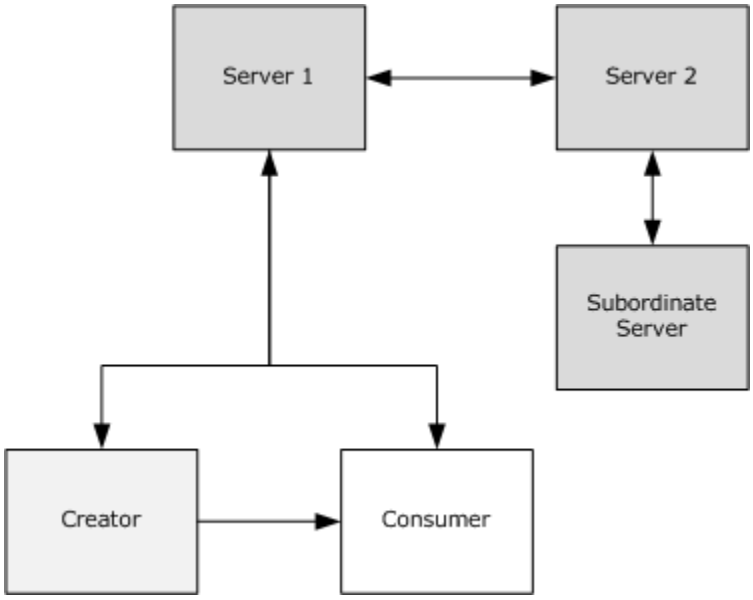

Figure 1: Roles in the Rights Management Services (RMS) system

For the creation and consumption of protected information (or content), the RMS system involves three active roles: the creator, the **consumer**, and the server. The creator and consumer are both roles of the RMS client. The interactions between the RMS client and the RMS server are specified in [MS-RMPR].

The RMS server has two primary responsibilities:

1. Issue **certificates** to end users for use during creation and consumption of protected content.
2. Make authorization decisions for access to protected content based on the policy that applies to that content, and issue **licenses** that record the authorization decision.

To issue licenses to end users, the RMS server must validate that the policy for the content grants access to the end user. The complexity for this validation can vary with the complexity of the surrounding network. For instance, distribution groups or other user groupings may require expansion; the directory topology may include partitions or other compartmentalization that restricts free movement of authorization and authentication data. These cases may necessitate the use of multiple RMS servers deployed in each partition to facilitate flow of information necessary for making authorization decisions that result in license issuance.

Some cases, such as when a specific department within an enterprise wants further control over information protection policies, require an RMS server deployed as a subordinate to an existing server. These situations require a specific trust to be established between RMS servers.

In these complicated deployments, RMS servers need to communicate with one another to share data. This communication uses the Rights Management Services (RMS): Server-to-Server Protocol defined by this specification.

1.3.1 Service Location Interface

RMS servers use the **Service Location** interface of the RMS: Server-to-Server Protocol to find the URLs for specific services that are provided by other RMS servers. This communication can be useful in two scenarios:

1. Finding the appropriate URLs for a client to use for bootstrapping.

2. Finding the Group Expansion interface on a remote server before making a cross-**forest** group expansion request.

Clients contact the server for a bootstrapping process so they may begin functioning in the RMS system. This bootstrapping process is defined in the RMS: Client-to-Server Protocol, as specified in [MS-RMPR]. To bootstrap a specific user, the RMS server needs to authenticate that user and determine the user's e-mail address by checking the directory. If the user's account resides in a partition of the directory that the RMS server cannot access, it cannot successfully bootstrap the user. A client starts the bootstrapping process by making a request for service locations to a specific RMS server. If that RMS server is not the appropriate server to bootstrap the client, the server can use the **Service Location** interface to find the URLs on the appropriate server, and then return them to the client.

The **Service Location** interface uses a SOAP-based protocol over HTTP. It exposes one request/response method: **FindServiceLocations**.

1.3.2 Sub-Enrollment Interface

RMS servers use the **Sub-Enrollment** interface of the RMS: Server-to-Server Protocol to bootstrap subordinate RMS servers.

As shown in Figure 1, RMS server can be deployed as a subordinate to another RMS server. A root RMS server grants a subordinate RMS server the right to perform only licensing tasks by issuing a subordinate **server licensor certificate (SLC)** from its own. For a subordinate RMS server, this process replaces the standard RMS server bootstrapping process defined in [MS-RMPR] section 3.3.<2>

The **Sub-Enrollment** interface uses a SOAP-based protocol over HTTP. It exposes one request/response method: **SubEnroll**.

1.3.3 Get Licensor Certificate Interface

RMS servers use the **Get Licensor Certificate** interface of the RMS: Server-to-Server Protocol to establish trust from a root server to a subordinate server.

When a subordinate RMS server is deployed, it needs to trust identities issued by the root RMS server. This is accomplished by trusting certificates that were issued by the root RMS server by trusting the root RMS server's public key. The subordinate server can use the **Get Licensor Certificate** interface to retrieve the server licensor certificate (SLC) of the main server that contains the appropriate public key.

The **Get Licensor Certificate** interface uses a SOAP-based protocol over HTTP. It exposes one request/response method: **GetLicensorCertificate**.

1.3.4 Group Expansion over SOAP Interface

RMS servers use the **Group Expansion over Remoting** interface of the RMS: Server-to-Server Protocol to determine group membership of authorized users across complex network directories.

Access policy on RMS protected content can specify individual users as well as distribution groups. When a consumer contacts the RMS server for authorization to access protected content, the server may need to consult the directory to determine if that user is a member of a group that is specified in the policy. If the group exists in a partition of the directory to which the RMS server does not have access, that RMS server needs to contact another server that does have appropriate

permissions. This server-to-server communication can use either the **Group Expansion over SOAP** interface or the **Group Expansion over Remoting** interface.

The **Group Expansion over SOAP** interface exposes one request/response method: **IsPrincipalMemberOf**.<3>

1.3.5 Group Expansion over Remoting Interface

The **Group Expansion over SOAP** interface performs the same function as the **Group Expansion over SOAP** interface, as defined in section 1.3.4, only it does so using a binary-formatted .NET Framework Remoting-based protocol over HTTP. It exposes one request/response method: **IsPrincipalMemberOf**.<4>

1.4 Relationship to Other Protocols

The **Service Location** interface, **Sub-Enrollment** interface, **Get Licensor Certificate** interface, and **Group Expansion over SOAP** interface all use a SOAP-based protocol that uses HTTP 1.1 as its transport.

The **Group Expansion over Remoting** interface uses a binary-formatted .NET Framework Remoting-based protocol over HTTP 1.1.

The following diagram shows the transport stack used by the RMS: Server-to-Server Protocol.

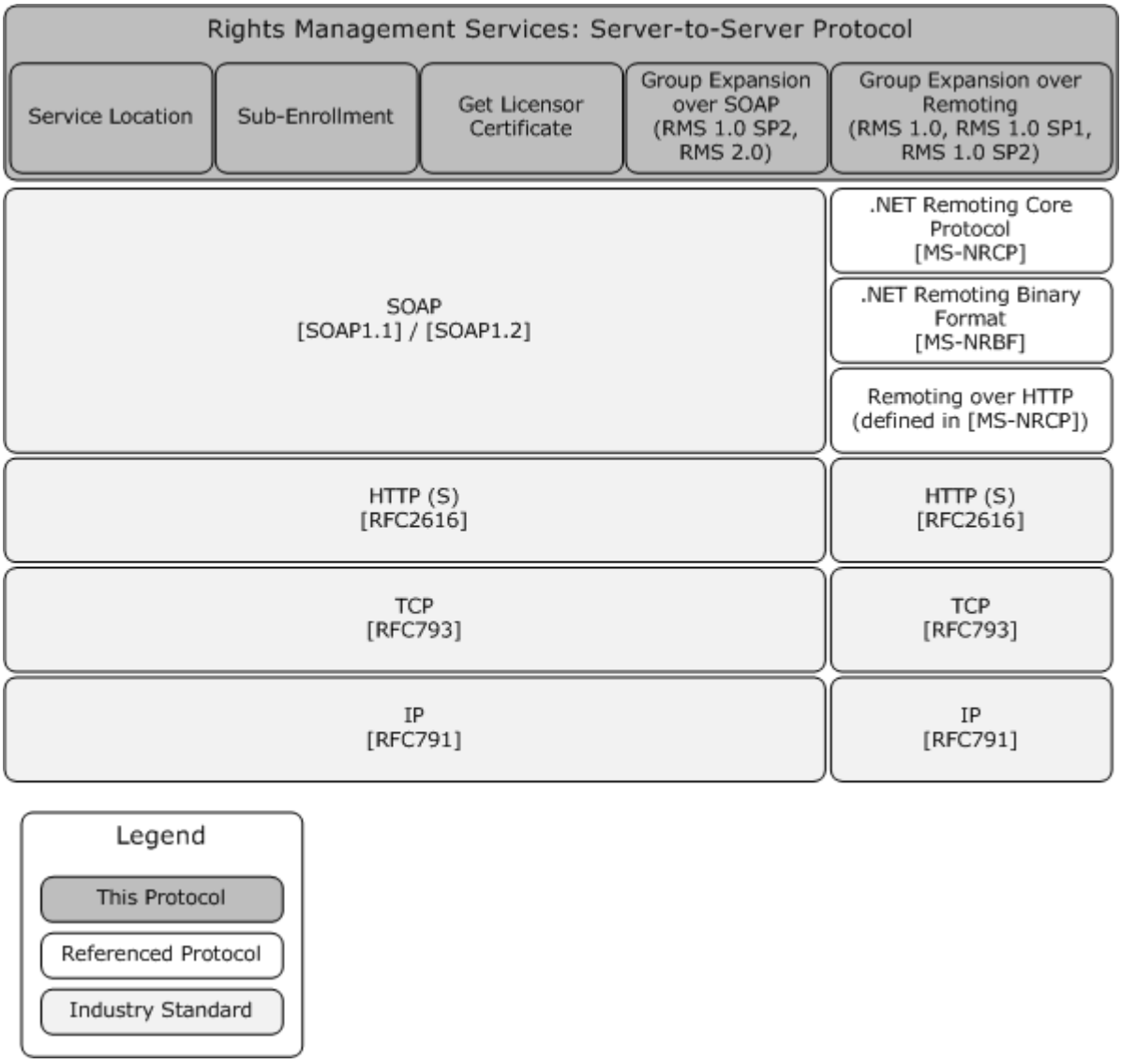

Figure 2: Transport stack for the RMS: Server-to-Server Protocol

1.4.1 Namespaces

The RMS: Server-to-Server Protocol defines the following Digital Rights Management (DRM) namespaces for use with the SOAP transport:

- http://microsoft.com/DRM/ServerService
- http://microsoft.com/DRM/SubEnrollmentService
- http://microsoft.com/DRM/GroupExpansionWebService

1.5 Prerequisites/Preconditions

To respond to **Service Location** requests, the server MUST be able to return the appropriate service locations for the services it either provides or references.

To respond to **Sub-Enrollment** and **Get Licensor Certificate** requests, the server MUST be properly bootstrapped and initialized with its own server licensor certificate (SLC) in place.

To respond to Group Expansion requests from either the **Group Expansion over Remoting** or **Group Expansion over SOAP** interfaces, the server MUST either have connectivity to the directory and permissions to query it, or the server MUST be able to verify group membership in some other way (for example, through cached directory information).

1.6 Applicability Statement

The RMS: Server-to-Server Protocol is used for communication between RMS servers when multiple server deployments are needed to support complex server topologies.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- **Supported Transports:** This protocol is implemented on top of HTTP, as specified in section 2.1.
- **Protocol Versions:** The Service Locations interface has only one version that is implemented using a SOAP-based protocol over HTTP. The **Group Expansion over Remoting** interface is used by RMS version 1.0 and RMS version 1.0 SP1. It is also used by RMS version 1.0 SP2 for backward compatibility when communicating with RMS version 1.0 servers or RMS version 1.0 SP1 servers. The **Group Expansion over SOAP** interface is used by RMS version 1.0 SP2 servers and RMS version 2.0 servers.<5>
- **Security and Authentication Methods:** This protocol passively supports Kerberos authentication over HTTP or HTTPS (as specified in [KERBKEY]) and NTLM authentication over HTTP or HTTPS (as specified in [NTLM]).
- **Localization:** There are no localization-dependent behaviors for the RMS: Server-to-Server Protocol.
- **Capability Negotiation:** The RMS: Server-to-Server Protocol supports limited capability negotiation via the VersionData type that is present on all SOAP-based protocol requests. On a request, the VersionData structure contains a MinimumVersion and MaximumVersion value, indicating the range of versions the client is capable of understanding. On a response, the VersionData structure contains a MinimumVersion and MaximumVersion that the server is capable of understanding. For .NET Remoting-based protocol requests, the requestor implementation is strongly bound to the version of the remoting interface being invoked.

1.8 Vendor-Extensible Fields

This protocol does not contain any vendor-extensible fields. All XML schema are considered non-extensible in the RMS: Server-to-Server Protocol.

1.9 Standards Assignments

The RMS: Server-to-Server Protocol uses the following standard XML namespaces:

- http://schemas.xmlsoap.org/wsdl/http/

- http://www.w3.org/2001/XMLSchema
- http://schemas.xmlsoap.org/wsdl/soap/
- http://schemas.xmlsoap.org/wsdl/soap12/
- http://schemas.xmlsoap.org/soap/encoding/
- http://schemas.xmlsoap.org/wsdl/

2 Messages

2.1 Transport

The RMS: Server-to-Server Protocol is composed of four SOAP-based interfaces and one .NET Framework Remoting-based interface:

- SOAP-based interfaces:
  - **Service Location**
  - **Sub-Enrollment**
  - **Get Licensor Certificate**
  - **Group Expansion over SOAP**
- .NET Framework Remoting-based interface:
  - **Group Expansion over Remoting**

Each SOAP-based interface MUST support SOAP (as specified in [SOAP1.1] or [SOAP1.2/1]) over HTTP (as specified in [RFC2616]) over TCP/IP. Each Web service SHOULD support HTTPS for securing communications.<6>

The **Group Expansion over Remoting** interface MUST use the .NET Remoting Core Protocol.

The interfaces MUST be exposed by the server at the following **endpoints**, starting from any base URL:

- **Service Location**: This interface, when implemented as specified in section 2.2.3, MUST be exposed at the following URLs:
  - [baseURL]/certification/server.asmx: FindServiceLocations
  - [baseURL]/licensing/server.asmx: FindServiceLocations
- **Sub-Enrollment**: This interface, when implemented as specified in section 2.2.4, MUST be exposed at the following URL:
  - [baseURL]/Certification/SubEnrollService.asmx: SubEnroll
- **Get Licensor Certificate**: This interface, when implemented as specified in section 2.2.5, MUST be exposed at the following URLs:
  - [baseURL]/certification/server.asmx: GetLicensorCertificate
  - [baseURL]/licensing/server.asmx: GetLicensorCertificate
- **Group Expansion over SOAP**: This interface, when implemented as specified in section 2.2.6, MUST be exposed at the following URL:
  - [baseURL]/groupexpansion/GroupExpansion.asmx: IsPrincipalMemberOf
- **Group Expansion over Remoting**: This interface, when implemented as specified in section 2.2.7, MUST be exposed at the following URL:
  - [baseURL]/ DrmRemote/DirectoryServices/DirectoryServices.rem

The **Group Expansion over Remoting** interface MUST support the .NET Remoting Core Protocol.

2.2 Message Syntax

2.2.1 Common Data Types

The following table shows the standard XML namespaces used within the RMS: Server-to-Server Protocol and the alias (prefix) use in the remainder of this section.

Alias (prefix)	XML namespace
http	http://schemas.xmlsoap.org/wsdl/http/
s	http://www.w3.org/2001/XMLSchema
soap	http://schemas.xmlsoap.org/wsdl/soap/
soap12	http://schemas.xmlsoap.org/wsdl/soap12/
soapenc	http://schemas.xmlsoap.org/soap/encoding/
wsdl	http://schemas.xmlsoap.org/wsdl/

2.2.2 Common SOAP Headers

All four SOAP-based interfaces in the RMS: Server-to-Server Protocol (that is, **Service Location**, **Sub-Enrollment**, **Get Licensor Certificate**, and **Group Expansion over SOAP**) use the same SOAP header for both requests and responses. The SOAP header for requests and responses to these interfaces MUST contain the VersionData element specified in section 2.2.2.1.

2.2.2.1 VersionData Complex Type

The VersionData complex type is used to represent the capability version of the requestor and the responder.

```
<xs:complexType name="VersionData">
  <xs:sequence>
    <xs:element name="MinimumVersion"
      type="string"
      minOccurs="0"
      maxOccurs="1"
     />
    <xs:element name="MaximumVersion"
      type="string"
      minOccurs="0"
      maxOccurs="1"
     />
  </xs:sequence>
</xs:complexType>
```

MinimumVersion: Specifies the lowest capability version supported. The version data in this type is represented by a literal string conforming to the format "a.b.c.d". Subversion value "a" is the most major component of the version, value "b" is the next most major, value "c" is the next most major, and "d" is the minor subversion value.

MaximumVersion: Specifies the highest capability version supported. The version data in this type is represented by a literal string conforming to the format "a.b.c.d". Subversion value "a" is the most major component of the version, value "b" is the next most major, value "c" is the next most major, and "d" is the minor subversion value.

2.2.3 Service Location Interface

An RMS server uses the **Service Location** interface to retrieve URLs for services that are either offered by or known to the responding server. The interface contains one method: **FindServiceLocations**.

2.2.3.1 FindServiceLocations

The **FindServiceLocations** method provides a mechanism for a requestor to retrieve a URL for a specified service that is either offered by the responding server or known to the responding server.

```
<wsdl:operation name="FindServiceLocations">
```

The SOAP operation is defined as given below.

```
<soap:operation
 soapAction=
  "http://microsoft.com/DRM/ServerService/FindServiceLocations"
 style="document" />
```

2.2.3.1.1 Request

The **FindServiceLocations** request submits a type of service for which a URL is being requested. Multiple service locations MAY be requested at one time.<7>

```
<wsdl:message name="FindServiceLocationsSoapIn">
  <wsdl:part name="parameters" element="tns:FindServiceLocations" />
</wsdl:message>
```

parameters: An element that contains the body of the SOAP request. A valid **FindServiceLocations** request MUST follow the schema below.

```
<s:element name="FindServiceLocations">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
        name="ServiceNames"
        type="tns:ArrayOfServiceLocationRequest"
      />
    </s:sequence>
  </s:complexType>
</s:element>
```

ServiceNames: An array that allows for multiple service locations to be requested at one time. For a valid request, the array MUST contain at least one request for a service location. The array MAY

contain more than one request for a service location. The array MUST follow the schema below in section 2.2.3.1.1.1.

2.2.3.1.1.1 ArrayOfServiceLocationRequest Complex Type

```
<s:complexType name="ArrayOfServiceLocationRequest">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
      name="ServiceLocationRequest" nillable="true"
      type="tns:ServiceLocationRequest" />
  </s:sequence>
</s:complexType>
```

ServiceLocationRequest: A single request for a service location that MUST request a service type. The request MUST follow the schema below in section 2.2.3.1.1.2.

2.2.3.1.1.2 ServiceLocationRequest Complex Type

```
<s:complexType name="ServiceLocationRequest">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1"
      name="Type"
      type="tns:ServiceType"
    />
  </s:sequence>
</s:complexType>
```

Type: The requested service type MUST be specified as one string the enumeration of all known service types below in section 2.2.3.1.1.3.

2.2.3.1.1.3 ServiceType Simple Type

```
<s:simpleType name="ServiceType">
  <s:restriction base="s:string">
    <s:enumeration value="EnrollmentService" />
    <s:enumeration value="LicensingService" />
    <s:enumeration value="PublishingService" />
    <s:enumeration value="CertificationService" />
    <s:enumeration value="ActivationService" />
    <s:enumeration value="PrecertificationService" />
    <s:enumeration value="ServerService" />
    <s:enumeration value="DrmRemoteDirectoryServices" />
    <s:enumeration value="GroupExpansionService" />
    <s:enumeration value="LicensingInternalService" />
    <s:enumeration value="CertificationInternalService" />
  </s:restriction>
</s:simpleType>
```

The following values are defined by the **ServiceType** simple type:

Value	Description
EnrollmentService	Identifies the enrollment service. MUST NOT be used with the RMS:

Value	Description
	Server-to-Server Protocol.
LicensingService	Identifies the licensing service (license.asmx).
PublishingService	Identifies the publishing service (publish.asmx). MUST NOT be used with the RMS: Server-to-Server Protocol.
CertificationService	Identifies the certification service (certification.asmx).
ActivationService	Identifies the activation service (activation.asmx). MUST NOT be used with the RMS: Server-to-Server Protocol.
PrecertificationService	Identifies the precertification service (precertification.asmx). MUST NOT be used with the RMS: Server-to-Server Protocol.
ServerService	Identifies the server service (server.asmx). MUST NOT be used with the RMS: Server-to-Server Protocol.
DrmRemoteDirectoryServices	Identifies the **Group Expansion over Remoting** interface.<8>
GroupExpansionService	Identifies the **Group Expansion over SOAP** interface.<9>
LicensingInternalService	Identifies the internal URL for the licensing service (license.asmx).<10>
CertificationInternalService	Identifies the internal URL for the certification service (certification.asmx).<11>

2.2.3.1.2 Response

The **FindServiceLocations** response returns a URL for each service that has been requested when a URL is available.

```
<wsdl:message name="FindServiceLocationsSoapOut">
  <wsdl:part name="parameters"
    element="tns:FindServiceLocationsResponse"
  />
</wsdl:message>
```

parameters: An element that contains the body of the SOAP response. A valid **FindServiceLocations** response MUST follow the schema below.

```
<s:element name="FindServiceLocationsResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
        name="FindServiceLocationsResult"
        type="tns:ArrayOfServiceLocationResponse"
      />
    </s:sequence>
  </s:complexType>
</s:element>
```

FindServiceLocationResult: An array that allows for multiple response elements to be returned at once. This array MUST contain the same number of elements as the array in the request. Each element of the array MUST contain a service location response, even if no URL was found. The array MUST follow the schema below in section 2.2.3.1.2.1.

2.2.3.1.2.1 ArrayOfServiceLocationResponse Complex Type

```
<s:complexType name="ArrayOfServiceLocationResponse">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
      name="ServiceLocationResponse" nillable="true"
      type="tns:ServiceLocationResponse"
    />
  </s:sequence>
</s:complexType>
```

ServiceLocationResponse: An individual response structure that contains the URL corresponding to a requested service type. Each service location response MUST contain a service type that matches the requested service type. The response includes the URL if found. The individual response structure MUST follow the schema below in section 2.2.3.1.2.2.

2.2.3.1.2.2 ServiceLocationResponse Complex Type

```
<s:complexType name="ServiceLocationResponse">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1"
      name="URL"
      type="s:string"
    />
    <s:element minOccurs="1" maxOccurs="1"
      name="Type"
      type="tns:ServiceType"
    />
  </s:sequence>
</s:complexType>
```

URL: A string that contains the URL of the service location.

Type: A string that identifies the service type from the ServiceType enumeration as defined above in section 2.2.3.1.1.3.

2.2.4 Sub-Enrollment Interface

An RMS server uses the **Sub-Enrollment** interface to generate and sign the server licensor certificate (SLC) so that it can be bootstrapped as a subordinate RMS server. Each RMS server must have a valid SLC to issue licenses and certificates. For SLC and server bootstrapping, see [MS-RMPR] sections 2.3.2, 2.3.3, and 3.1.3.

This interface contains one method: **SubEnroll**.

2.2.4.1 SubEnroll

A subordinate RMS server generates its own asymmetric key pair. It then uses the **SubEnroll** method to request that its public key be contained within an SLC chain that it can use to issue licenses to protected content.

```
<wsdl:operation name="SubEnroll">
```

The SOAP operation is defined as given below.

```
<soap:operation
 soapAction="http://microsoft.com/DRM/SubEnrollmentService/SubEnroll"
 style="document" />
```

2.2.4.1.1 Request

The SubEnroll request submits a public key and attributes of the RMS server that is making the request.

```
<wsdl:message name="SubEnrollSoapIn">
  <wsdl:part name="parameters" element="tns:SubEnroll" />
</wsdl:message>
```

parameters: An element that contains the body of the SOAP request. The body MUST contain exactly one **SubEnroll** element and MUST follow the schema below.

```
<s:element name="SubEnroll">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1"
        name="oInput"
        type="tns:SubEnrollParameters"
      />
    </s:sequence>
  </s:complexType>
</s:element>
```

oInput: An object that contains the parameters of the enrollment request. The parameters MUST include a public key and attributes of the requestor. The object MUST follow the schema below in section 2.2.4.1.1.1.

2.2.4.1.1.1 SubEnrollParameters ComplexType

```
<s:complexType name="SubEnrollParameters">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1"
      name="CertificatePublicKey"
      type="tns:EnrolleeCertificatePublicKey"
    />
    <s:element minOccurs="1" maxOccurs="1"
      name="EnrolleeInformation"
```

```
      type="tns:EnrolleeServerInformation"
    />
  </s:sequence>
</s:complexType>
```

CertificatePublicKey: A structure that contains the requestor's public key and GUID. It MUST follow the EnrolleeCertificatePublicKey schema in section 2.2.4.1.1.2 below.

EnrolleeInformation: A structure that contains attributes of the requestor. It MUST follow the EnrolleeServerInformation schema in section 2.2.4.1.1.3 below.

2.2.4.1.1.2 EnrolleeCertificatePublicKey Complex Type

```
<s:complexType name="EnrolleeCertificatePublicKey">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1"
      name="aPublicKeyBytes"
      type="s:base64Binary"
    />
    <s:element minOccurs="1" maxOccurs="1"
      name="Guid"
      type="s1:guid"
    />
  </s:sequence>
</s:complexType>
```

aPublicKeyBytes: Contains the requestor's public key represented as a base-64 encoded string.<12>

Guid: Contains a GUID that identifies the requesting server among all other servers in the RMS system.

2.2.4.1.1.3 EnrolleeServerInformation Complex Type

```
<s:complexType name="EnrolleeServerInformation">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1"
      name="SKU"
      type="s:string"
    />
    <s:element minOccurs="0" maxOccurs="1"
      name="Version"
      type="s:string"
    />
    <s:element minOccurs="0" maxOccurs="1"
      name="Name"
      type="s:string"
    />
    <s:element minOccurs="0" maxOccurs="1"
      name="URL"
      type="s:string"
    />
  </s:sequence>
</s:complexType>
```

SKU: A string describing the SKU information of the requesting server.

Version: A string describing the version information of the requesting server.

Name: A string describing the friendly name for the requesting server.

URL: A string that contains the URL of the requesting server.

2.2.4.1.2 Response

The **SubEnroll** response returns a signed SLC chain.

```
<wsdl:message name="SubEnrollSoapOut">
  <wsdl:part name="parameters" element="tns:SubEnrollResponse" />
</wsdl:message>
```

parameters: An element that contains the body of the SOAP response including the SLC chain. It MUST follow the schema below.

```
<s:element name="SubEnrollResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1"
        name="SubEnrollResult"
        type="tns:SubEnrollResponse"
      />
    </s:sequence>
  </s:complexType>
</s:element>
```

SubEnrollResult: A structure that contains the resulting SLC chain and MUST follow the schema below in section 2.2.4.1.2.1.

2.2.4.1.2.1 SubEnrollResponse Complex Type

```
<s:complexType name="SubEnrollResponse">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1"
      name="LicensorCertificateChain"
      type="tns:ArrayOfString"
    />
  </s:sequence>
</s:complexType>
```

LicensorCertificateChain: An array of strings, with each element containing one node of the resulting SLC chain. It MUST follow the schema below in section 2.2.4.1.2.2.

2.2.4.1.2.2 ArrayOfString Complex Type

```
<s:complexType name="ArrayOfString">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
      name="string" nillable="true"
```

```
        type="s:string"
      />
    </s:sequence>
  </s:complexType>
```

string: A string that contains a node of a certificate chain.

2.2.5 Get Licensor Certificate Interface

An RMS server uses the **Get Licensor Certificate** interface to acquire the SLC of the responder. The interface contains one method: **GetLicensorCertificate**.

2.2.5.1 GetLicensorCertificate

This method provides a mechanism for a requestor to acquire the SLC chain of the responder. The format of the SLC chain is as defined in [MS-RMPR] section 2.3.

```
<wsdl:operation name="GetLicensorCertificate">
```

The SOAP operation is defined as given below.

```
<soap:operation
 soapAction=
  "http://microsoft.com/DRM/ServerService/GetLicensorCertificate"
 style="document" />
```

2.2.5.1.1 Request

The **GetLicensorCertificate** request submits no inputs.

```
<wsdl:message name="GetLicensorCertificateSoapIn">
  <wsdl:part name="parameters" element="tns:GetLicensorCertificate" />
</wsdl:message>
```

parameters: An element that contains the body of the SOAP request. The body MUST contain exactly one **GetLicensorCertificate** element and MUST follow the schema below.

```
<s:element name="GetLicensorCertificate">
  <s:complexType />
</s:element>
```

2.2.5.1.2 Response

The **GetLicensorCertificate** response returns the requestor's SLC chain.

```
<wsdl:message name="GetLicensorCertificateSoapOut">
  <wsdl:part name="parameters"
    element="tns:GetLicensorCertificateResponse"
  />
```

```
</wsdl:message>
```

parameters: An element that contains the body of the SOAP response including the responder's SLC chain. It MUST follow the schema below.

```
<s:element name="GetLicensorCertificateResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
        name="GetLicensorCertificateResult"
        type="tns:LicensorCertChain"
      />
    </s:sequence>
  </s:complexType>
</s:element>
```

GetLicensorCertificateResult: A structure containing the responder's SLC chain. It MUST follow the schema below in section 2.2.5.1.2.1.

2.2.5.1.2.1 LicensorCertChain Complex Type

```
<s:complexType name="LicensorCertChain">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1"
      name="CertificateChain"
      type="tns:ArrayOfXmlNode"
    />
  </s:sequence>
</s:complexType>
```

CertificateChain: Contains the responder's SLC chain as an array of XML nodes. It MUST follow the schema below in section 2.2.5.1.2.2.

2.2.5.1.2.2 ArrayOfXmlNode Complex Type

```
<s:complexType name="ArrayOfXmlNode">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
      name="Certificate" nillable="true">
      <s:complexType mixed="true">
        <s:sequence>
          <s:any />
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:sequence>
</s:complexType>
```

Certificate: XML that contains the SLC certificate chain.

2.2.6 Group Expansion over SOAP Interface

An RMS server uses the **Group Expansion over SOAP** interface to verify group membership of a specific user with another RMS server. The interface contains one method: **IsPrincipalMemberOf**.

2.2.6.1 IsPrincipalMemberOf

The IsPrincipalMemberOf method provides a mechanism for a requestor to verify with a responder whether a specific user is currently a member of specific groups that the requestor cannot expand by contacting the directory itself.

It is possible that a requested group contains a subgroup in another forest, causing the responder to make a new IsPrincipalMemberOf request to another server before it can respond to the original requestor. To prevent infinite loops or unacceptably long response times, the request specifies a number of servers that have been involved in servicing this group expansion request so far.

```
<wsdl:operation name="IsPrincipalMemberOf">
```

The SOAP operation is defined as given below.

```
<soap:operation
 soapAction=
  "http://microsoft.com/DRM/GroupExpansionWebService/IsPrincipalMemberOf"
 style="document"/>
```

2.2.6.1.1 Request

The **IsPrincipalMemberOf** request submits information about the user, one or more groups to be checked, and a count of the number of servers that have been involved in servicing this group expansion request so far.

```
<wsdl:message name="IsPrincipalMemberOfSoapIn">
  <wsdl:part name="parameters" element="tns:IsPrincipalMemberOf" />
</wsdl:message>
```

parameters: An element that contains the body of the SOAP request. The body MUST contain exactly one **IsPrincipalMemberOf** element and MUST follow the schema below.

```
<s:element name="IsPrincipalMemberOf">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
        name="principalName"
        type="s:string"
      />
      <s:element minOccurs="0" maxOccurs="1"
        name="principalCrossForest"
        type="s:string"
      />
      <s:element minOccurs="0" maxOccurs="1"
        name="targetGroups"
        type="tns:ArrayOfString"
      />
```

```
        <s:element minOccurs="1" maxOccurs="1"
          name="crossForestCallsSoFar"
          type="s:int"
        />
      </s:sequence>
    </s:complexType>
  </s:element>
```

principalName: A string that contains the e-mail address of the user whose group membership is to be verified.

principalCrossForest: A string that contains the e-mail address of the user whose group membership is to be verified.

targetGroups: An array of strings that contains one or more e-mail addresses representing the groups for which membership is to be checked. MUST follow the schema in section 2.2.6.1.1.1 below.

crossForestCallsSoFar: An integer that represents the number of servers that have been involved in servicing this group expansion request so far.

2.2.6.1.1.1 ArrayOfString Complex Type

```
<s:complexType name="ArrayOfString">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded"
      name="string" nillable="true"
      type="s:string"
    />
  </s:sequence>
</s:complexType>
```

string: A string that contains the e-mail address contact for a group for which membership is to be checked.

2.2.6.1.2 Response

The **IsPrincipalMemberOf** response returns a Boolean value that either confirms or denies group membership.

```
<wsdl:message name="IsPrincipalMemberOfSoapOut">
  <wsdl:part name="parameters"
    element="tns:IsPrincipalMemberOfResponse"
  />
</wsdl:message>
```

parameters: An element that contains the body of the SOAP response containing the group membership status. It MUST contain an SLC chain and it MUST follow the schema below.

```
<s:element name="IsPrincipalMemberOfResponse">
  <s:complexType>
    <s:sequence>
```

```
        <s:element minOccurs="1" maxOccurs="1"
          name="IsPrincipalMemberOfResult"
          type="s:boolean"
        />
      </s:sequence>
    </s:complexType>
  </s:element>
```

IsPrincipalMemberOfResult: A Boolean value that either confirms or denies group membership.

2.2.7 Group Expansion over Remoting Interface

An RMS server uses the **Group Expansion over Remoting** interface to verify group membership of a specific user with another RMS server. The interface contains one method: **IsPrincipalMemberOf**.

2.2.7.1 IsPrincipalMemberOf

The **IsPrincipalMemberOf** method provides a mechanism for a requestor to verify with a responder whether a specific user is currently a member of specific groups that the requestor cannot expand by contacting the directory itself. **IsPrincipalMemberOf** returns positive if the user is a member of at least one of the specified groups and returns false otherwise.

It is possible that a requested group contains a subgroup in another forest, causing the responder to make a new **IsPrincipalMemberOf** request to another server before it can respond to the original requestor. To prevent infinite loops or unacceptably long response times, the request specifies a number of servers that have been involved in servicing this group expansion request so far.

```
namespace Microsoft.DigitalRightsManagement.DirectoryServices
{
  interface IRemoteActiveDirectoryServices
  {
    Bool IsPrincipalMemberOf(String principal,
    String principalCrossForest, String[] targetGroups,
    Int32 crossForestCallsSoFar,
    out Microsoft.DigitalRightsManagement.DirectoryServices.Principal
    resultPrincipal);
  }
}
```

2.2.7.1.1 Request

The **IsPrincipalMemberOf** request submits information about the user, one or more groups to be checked, and a count of the number of servers that have been involved in servicing this group expansion request so far.

```
Bool IsPrincipalMemberOf(
   String principal,
   String principalCrossForest,
   String[] targetGroups,
   Int32 crossForestCallsSoFar,
   out Microsoft.DigitalRightsManagement.DirectoryServices.Principal
      resultPrincipal);
```

principal: A string that contains the e-mail address of the user whose group membership is to be verified.

principalCrossForest: A string that contains the e-mail address of the user whose group membership is to be verified.

targetGroups: An array of strings that contain the e-mail address contact for one or more groups for which membership is to be checked.

crossForestCallsSoFar: A 32-bit integer that represents the number of servers that have been involved in servicing this group expansion request so far.

2.2.7.1.2 Response

The **IsPrincipalMemberOf** response returns a Boolean value that either confirms or denies group membership along with a result.

```
Bool IsPrincipalMemberOf(
   String principal,
   String principalCrossForest,
   String[] targetGroups,
   Int32 crossForestCallsSoFar,
   out Microsoft.DigitalRightsManagement.DirectoryServices.Principal
      resultPrincipal);
```

Bool IsPrincipalMemberOf: A Boolean value that either confirms or denies group membership.

resultPrincipal: A structure that contains information about the request and response. MUST be null if none of the requested groups is found. Otherwise MUST be of a Principal class as defined below in section 2.2.7.1.2.1.

2.2.7.1.2.1 Principal Class

```
class Principal
{
  System.Collections.ListDictionary      _PrincipalIdentifiers;
  System.Collections.Hashtable           _GroupMembership;
  System.Collections.ListDictionary      _ForeignMembers;
  System.Collections.ListDictionary      _parsingDictionary;
  System.Collections.StringCollection    _ContainerObjectGuids;
  String                                 _strObjectGuid;
  String                                 _strOriginationForest;
  Microsoft.DigitalRightsManagement.DirectoryServices.ExplicitParseEnum
                                         _explicitParse;
  Bool                                   _exists;
}
```

_PrincipalIdentifiers: A ListDictionary as defined below in section 2.2.7.1.2.2, containing all identifiers for the specified user including a Security Identifier (SID) and all e-mail addresses and proxy addresses. A SID is specified as a string prefixed with "id=" while an e-mail address is specified as a string prefixed with "mail=". Identifiers MUST be represented as strings in the dictionary. Values in the ListDictionary are unused, but MUST be of type Boolean and set to true. _PricipalIdentifiers MUST be null if the principal cannot be found.

_GroupMembership: A Hashtable as defined below in section 2.2.7.1.2.4, containing group membership identifiers as keys. Values in the Hashtable are unused, but MUST be of type Boolean and set to true. _GroupMembership MUST be null if the principal cannot be found.

_ForeignMembers: A ListDictionary, as defined below in section 2.2.7.1.2.2, mapping forest names to contacts from those forests. Contact identifiers are represented as strings and values describe the forest name, represented as strings. ForeignMembers MUST be null if the principal cannot be found.

_parsingDictionary: Must be null.

_ContainerObjectGuids: A collection of GUIDs of all container objects that were queried in the directory, expressed as a StringCollection as defined below in section 2.2.7.1.2.5. Each GUID is expressed as a string.

_strObjectGuid: A GUID for the user's object from the directory, represented as a string.

_strOriginationForest: A string identifying the originating forest for the principal.

_explicitParse: A value from the ExplicitParseEnum that describes how the responder parsed the principal as defined below in section 2.2.7.1.2.7.

_exists: A Boolean value that specifies whether the requested principal exists in the directory.

2.2.7.1.2.2 ListDictionary Class

A ListDictionary is a list of DictionaryNodes of a specified length in which each node points to the next.

```
class ListDictionary
{
  System.Collections.ListDictionary+DictionaryNode    head;
  Int32                                               count;
  Int32                                               version;
}
```

head: The head node of the list of type ListDictionary+DictionaryNode as defined below in section 2.2.7.1.2.3.

count: A 32-bit integer that specifies the size of the list, starting with the head node as node 1.

version: A 32-bit integer that specifies a version for the list, representing the number of modifications that have been made to the list.

2.2.7.1.2.3 ListDictionary+DictionaryNode Class

A ListDictionary+DictionaryNode is a node in a ListDictionary.

```
class ListDictionary+DictionaryNode
{
  String                                              key;
  System.Object                                       value;
  System.Collections.ListDictionary+DictionaryNode    next;
}
```

key: A string containing the data of the list entry, such as an e-mail address.

value: An object. Type and usage are defined above in section 2.2.7.1.2.1.

next: The next value in the list. Set to null to terminate the list.

2.2.7.1.2.4 Hashtable Class

A standard hashtable class representing a dictionary of associated data keys and values with constant lookup time.

```
class Hashtable
{
  Int32       LoadFactor;
  Int32       Version;
  Int32       HashSize;
  String[]    Keys;
  Bool[]      Values;
}
```

LoadFactor: A 32-bit integer that indicates the maximum ratio of hash table entries to hash table buckets.

Version: A 32-bit integer that specifies a version for the hash table, representing the number of modifications that have been made to the list.

HashSize: A 32-bit integer used to compute the hash function.

Keys: An array of strings containing the data keys for the hash table.

Values: An array of Boolean values for the hash table.

2.2.7.1.2.5 StringCollection Class

A collection of strings.

```
class StringCollection
{
  System.Collections.ArrayList    data;
}
```

data: A collection of data, expressed as an ArrayList (section 2.2.7.1.2.6).

2.2.7.1.2.6 ArrayList Class

An array of objects.

```
class ArrayList
{
  System.Object[]     _items;
  Int32               _size;
  Int32               _version;
}
```

_items: An array of objects containing data.

_size: A 32-bit integer describing the size of the array.

_version: A 32-bit integer that specifies a version for the array, representing the number of modifications that have been made to the list.

2.2.7.1.2.7 ExplicitParseEnum Enum

An enumeration of possible methods for how the principal was parsed.

```
enum ExplicitParseEnum
{
  None = 0,
  PrimaryMail = 1,
  RFC822 = 2
}
```

None: No restrictions.

PrimaryMail: Check the user's primary e-mail address only.

RFC822: Check RFC822 addresses only.

2.2.8 Common Fault Codes

For SOAP-based interfaces, the RMS: Server-to-Server Protocol allows a server to notify a requestor of application-level faults by generating SOAP faults (as specified in [SOAP1.1] section 4.4). In the SOAP fault, the <faultcode> element contains the type of exception being thrown. The <faultstring> element contains the text of the exception being thrown.

For .NET Framework Remoting-based interfaces, the RMS: Server-to-Server Protocol notifies the requestor of the same set of application-level faults through .NET Framework Remoting.

The following table summarizes the exceptions that the responding server can return to the requesting server.

Exception	Description
ArgumentException	.NET Framework exception.
ArgumentOutOfRangeException	.NET Framework exception.
ArgumentNullException	.NET Framework exception.
FormatException	.NET Framework exception.
UnauthorizedAccessException	Access is unauthorized.
ADEntrySearchFailedException	Failed to find an entry in **Active Directory (AD)**.
DRMSArgumentException	An argument exception occurred. See the inner exception.
MalformedDataVersionException	A client request contained an invalid version number that cannot be processed.

Exception	Description
UnsupportedDataVersionException	The data version the client requested is not supported. The server cannot process the request.

3 Protocol Details

The Rights Management Services (RMS): Server-to-Server Protocol operates between two RMS servers. The initiator or requestor is the client for the protocol, and the responder is the server for the protocol. The protocol allows for stateless server operation.

The client side of this protocol is simply a pass-through. That is, there are no additional timers or other state required on the client side of this protocol. Calls made by the higher-layer protocol or application are passed directly to the transport, and the results returned by the transport are passed directly back to the higher-layer protocol or application.

3.1 Common SOAP Headers

All four SOAP-based interfaces in the RMS: Server-to-Server Protocol (that is, Service Location, Sub-Enrollment, Get Licensor Certificate, and Group Expansion over SOAP) use the same SOAP header for both requests and responses. The SOAP header for requests and responses to these interfaces MUST contain the VersionData element specified in section 2.2.2.1.

Request:

When a request is made, the requestor MUST specify the lowest capability version it can support as the MinimumVersion parameter. The client MUST specify the highest capability version it can support as the MaximumVersion parameter. The client MUST make the request in accordance with the MaximumVersion capability version.

Parameter	Description
MinimumVersion	MUST specify the lowest capability version supported by the requestor.
MaximumVersion	MUST specify the highest capability version supported by the requestor.

Data Processing:

When a responder receives a request, it MUST compare its own capability version to the capability version range presented by the requestor. The responder MUST assume that the requestor always makes a maximum-version request. The responder MUST reject the request with an error if its highest capability version is lower than the MaximumVersion specified by the requestor.

Response:

When responding to a requestor, including when responding with an error, the responder MUST specify the lowest capability version it can support as the MinimumVersion parameter. The responder MUST specify the highest capability version it can support as the MaximumVersion parameter. If the responder's maximum capability version is lower than the requestor's maximum capability version, the requestor SHOULD resend its request and alter its request to conform to the capability version range specified by the responder.<13>

3.2 Service Location Interface Server Details

A requesting RMS server uses the **Service Location** interface to discover the URLs of services offered by a responding RMS server. These URLs MAY be passed back to a client as a form of redirection, or they MAY be used by the requesting server to make another request of the responding RMS server.

The **Service Location** interface provides one method: **FindServiceLocations**.

3.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document. The conceptual data can be implemented by using a variety of techniques. Any data structure that stores the conceptual data MAY be used in the implementation.

In order to properly implement the **Service Location** interface, the responder MUST be able to provide the URLs it uses.

3.2.2 Timers

The **Service Location** interface has no timers.

3.2.3 Initialization

No specific initialization steps are required for the **Service Location** interface.

3.2.4 Message Processing Events and Sequencing Rules

3.2.4.1 FindServiceLocations

This method provides a mechanism for a requestor to retrieve a URL for a specified service that is either offered by the responding server or known to the responding server.

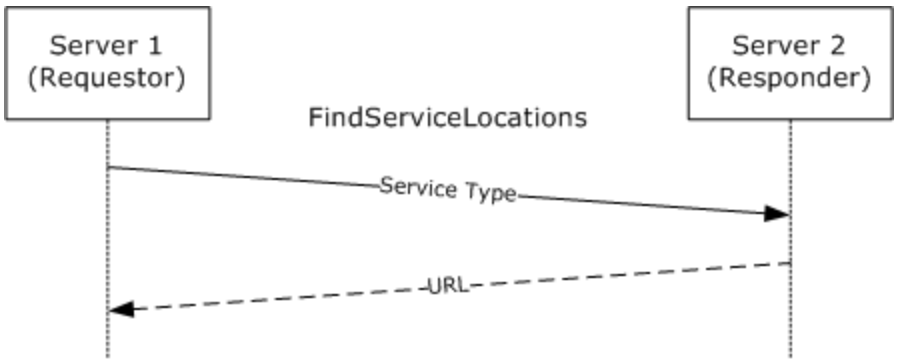

Figure 3: FindServiceLocations message sequence diagram

Request Validation:

The responding server MUST validate the input parameters upon receiving a FindServiceLocations request. The **FindServiceLocations** request MUST follow the schema specified above in section 2.2.3.1.1.

Parameter	Description
ServiceType	MUST be a valid service type from the following list: LicensingService, CertificationService, DrmRemoteDirectoryServices, GroupExpansionService, LicensingInternalService, CertificationInternalService. Other service types are not supported. A single request MAY request URLs for multiple service types.

Data Processing:

For a successful request, the responding server MUST return the appropriate service location URL. For an unsuccessful request, the server MUST return a fault code. The following table describes the URL expected for each service type.

Service Type	Description
LicensingService	The URL of the responding server's licensing service (license.asmx).
CertificationService	The URL of the responding server's certification service (certification.asmx).
DrmRemoteDirectoryServices	The URL of the responding server's **Group Expansion over Remoting** interface.<14>
GroupExpansionService	The URL of the responding server's **Group Expansion over SOAP** interface.<15>
LicensingInternalService	The internal URL of the responding server's the licensing service (license.asmx).<16>
CertificationInternalService	The internal URL of the responding server's certification service (certification.asmx).<17>

Response:

A successful **FindServiceLocations** response MUST follow the schema specified above in section 2.2.3.1.2. If multiple service types were requested, a successful response MUST include the same set of service types with corresponding URLs. If a specific URL is not available, the URL MUST be null but the service type MUST still be included. For an unsuccessful request, the server MUST return a fault code. This method throws only Common Fault Codes for the RMS: Server-to-Server Protocol as specified above in section 2.2.8.

Request template:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <VersionData xmlns="http://microsoft.com/DRM/ServerService">
      <MinimumVersion>string</MinimumVersion>
      <MaximumVersion>string</MaximumVersion>
    </VersionData>
  </soap:Header>
  <soap:Body>
    <FindServiceLocations xmlns="http://microsoft.com/DRM/ServerService">
      <ServiceNames>
        <ServiceLocationRequest>
          <Type>EnrollmentService or
                LicensingService or
                PublishingService or
                CertificationService or
                ActivationService or
                PrecertificationService or
                ServerService or
                DrmRemoteDirectoryServices or
                GroupExpansionService or
                LicensingInternalService or
```

```
            CertificationInternalService</Type>
        </ServiceLocationRequest>
        <ServiceLocationRequest>
          <Type>EnrollmentService or
            LicensingService or
            PublishingService or
            CertificationService or
            ActivationService or
            PrecertificationService or
            ServerService or
            DrmRemoteDirectoryServices or
            GroupExpansionService or
            LicensingInternalService or
            CertificationInternalService</Type>
        </ServiceLocationRequest>
      </ServiceNames>
    </FindServiceLocations>
  </soap:Body>
</soap:Envelope>
```

Response template:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <VersionData xmlns="http://microsoft.com/DRM/ServerService">
      <MinimumVersion>string</MinimumVersion>
      <MaximumVersion>string</MaximumVersion>
    </VersionData>
  </soap:Header>
  <soap:Body>
    <FindServiceLocationsResponse
     xmlns="http://microsoft.com/DRM/ServerService">
      <FindServiceLocationsResult>
        <ServiceLocationResponse>
          <URL>string</URL>
          <Type>EnrollmentService or
                LicensingService or
                PublishingService or
                CertificationService or
                ActivationService or
                PrecertificationService or
                ServerService or
                DrmRemoteDirectoryServices or
                GroupExpansionService</Type>
        </ServiceLocationResponse>
        <ServiceLocationResponse>
          <URL>string</URL>
          <Type>EnrollmentService or
                LicensingService or
                PublishingService or
                CertificationService or
                ActivationService or
                PrecertificationService or
                ServerService or
                DrmRemoteDirectoryServices or
```

```
                GroupExpansionService</Type>
          </ServiceLocationResponse>
        </FindServiceLocationsResult>
      </FindServiceLocationsResponse>
    </soap:Body>
  </soap:Envelope>
```

3.2.5 Timer Events

The **Service Location** interface has no timer events.

3.2.6 Other Local Events

The **Service Location** interface has no other local events.

3.3 Sub-Enrollment Interface Server Details

The **Sub-Enrollment** interface provides a mechanism for a requesting RMS server to be bootstrapped as a subordinate of a responding RMS server. Following a successful bootstrapping process, the requestor will have the authority to perform licensing tasks but not certification tasks. Licensing and certification are defined in [MS-RMPR] sections 3.1.7.9 and 3.1.7.2, respectively.

The **Sub-Enrollment** interface provides one method: **SubEnroll**.

3.3.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document. The conceptual data can be implemented by using a variety of techniques. Any data structure that stores the conceptual data MAY be used in the implementation.

SLC chain: An XrML 1.2 certificate chain that signs the RMS server's public key into the appropriate certificate hierarchy. The SLC format is specified in RMS: Client-Server Protocol Specification ([MS-RMPR] section 2.3).

Server key pair: An asymmetric key pair used for encryption and signing on the server.<18>

3.3.2 Timers

The **Sub-Enrollment** interface has no timers.

3.3.3 Initialization

The requesting RMS server MUST generate its asymmetric key pair before calling the **Sub-Enrollment** interface on a responding server.

The responding RMS server MUST be bootstrapped in order for the **Sub-Enrollment** interface to function. RMS server bootstrapping is defined in [MS-RMPR] section 3.1.3.

3.3.4 Message Processing Events and Sequencing Rules

3.3.4.1 SubEnroll

During the **SubEnroll** request, the requestor submits its public key along with metadata about itself in order to receive a signed SLC that grants it the right to issue licenses.

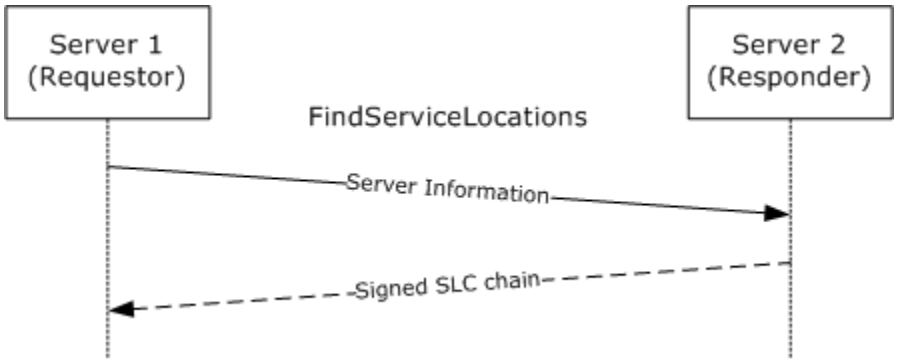

Figure 4: SubEnroll message sequence diagram

Request Validation:

The responding server MUST validate the input parameters upon receiving a SubEnroll request. The **SubEnroll** request MUST follow the schema specified above in section 2.2.4.1.1.

Parameter	Description
aPublicKeyBytes	MUST contain the requestor's public key, represented as a base-64 encoded string. This key will be the public key that is issued in the resulting SLC.<19>
Guid	MUST contain a GUID that will be used to identify the public key inside the resulting SLC. MUST be formatted as a string.
SKU	MUST be a string. SHOULD contain SKU information of the requesting server.
Version	MUST be a string. SHOULD describe the version information of the requesting server.
URL	MUST be a string that contains the URL of the requesting server.

Data Processing:

For a successful request, the responding server MUST generate and return a signed SLC chain. The leaf-node SLC MUST contain the public key that was submitted in the request and associate the SKU, version, and URL with that key. The responder's own SLC chain MUST be appended to the SLC that is generated for the requestor. The SLC format is specified in [MS-RMPR] section 2.3.3. The SLC chain is specified in [MS-RMPR] section 2.3.2.

Response:

A successful **SubEnroll** response MUST follow the schema specified above in section 2.2.4.1.2. A successful response MUST return the SLC chain that was generated for the requestor. For an unsuccessful request, the server MUST return a fault code. This method throws only Common Fault Codes for the RMS: Server-to-Server Protocol.

Request template:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <VersionData
     xmlns="http://microsoft.com/DRM/SubEnrollmentService">
      <MinimumVersion>string</MinimumVersion>
      <MaximumVersion>string</MaximumVersion>
    </VersionData>
  </soap:Header>
  <soap:Body>
    <SubEnroll xmlns="http://microsoft.com/DRM/SubEnrollmentService">
      <oInput>
        <CertificatePublicKey>
          <aPublicKeyBytes>base64Binary</aPublicKeyBytes>
          <Guid>guid</Guid>
        </CertificatePublicKey>
        <EnrolleeInformation>
          <SKU>string</SKU>
          <Version>string</Version>
          <Name>string</Name>
          <URL>string</URL>
        </EnrolleeInformation>
      </oInput>
    </SubEnroll>
  </soap:Body>
</soap:Envelope>
```

Response template:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <VersionData xmlns="http://microsoft.com/DRM/SubEnrollmentService">
      <MinimumVersion>string</MinimumVersion>
      <MaximumVersion>string</MaximumVersion>
    </VersionData>
  </soap:Header>
  <soap:Body>
    <SubEnrollResponse
     xmlns="http://microsoft.com/DRM/SubEnrollmentService">
      <SubEnrollResult>
        <LicensorCertificateChain>
          <string>string</string>
          <string>string</string>
        </LicensorCertificateChain>
      </SubEnrollResult>
    </SubEnrollResponse>
  </soap:Body>
</soap:Envelope>
```

3.3.5 Timer Events

The **Sub-Enrollment** interface has no timer events.

3.3.6 Other Local Events

The **Sub-Enrollment** interface has no other local events.

3.4 Get Licensor Certificate Interface Server Details

When an RMS server becomes a subordinate of another RMS server, it does not issue its own **RMS Account Certificates (RACs)**; instead, it trusts RMS Account Certificates (RACs) issued by the root RMS server. To trust RMS Account Certificates (RACs) issued by the root RMS server, the subordinate RMS server needs to know the public key of the root RMS server, which is contained in the root RMS server's server licensor certificate (SLC).

A subordinate RMS server uses the **Get Licensor Certificate** interface to acquire the root RMS server's server licensor certificate (SLC) chain. The **Get Licensor Certificate** interface provides one method: **GetLicensorCertificate**.

3.4.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document. The conceptual data can be implemented by using a variety of techniques. Any data structure that stores the conceptual data MAY be used in the implementation.

SLC chain: An XrML 1.2 certificate chain that signs the RMS server's public key into the appropriate certificate hierarchy. The SLC format is specified in [MS-RMPR] section 2.3.

3.4.2 Timers

The **Get Licensor Certificate** interface has no timers.

3.4.3 Initialization

The responding RMS server MUST be bootstrapped in order for the **Sub-Enrollment** interface to function. RMS server bootstrapping is defined in [MS-RMPR] section 3.1.3.

3.4.4 Message Processing Events and Sequencing Rules

3.4.4.1 GetLicensorCertificate

The **GetLicensorCertificate** method takes no input parameters and returns the responder's SLC.

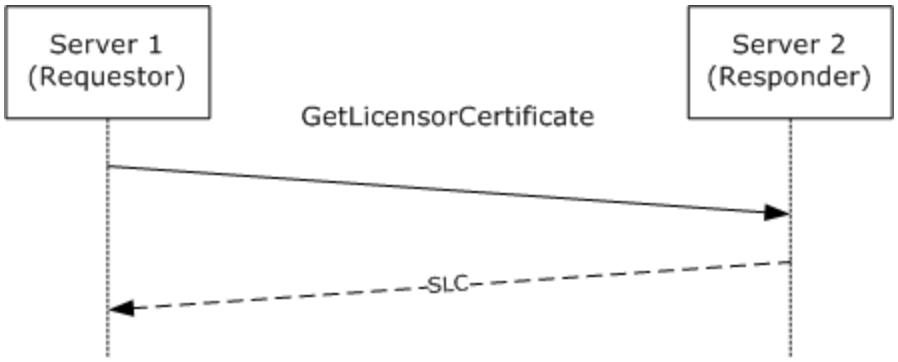

Figure 5: GetLicensorCertificate message sequence diagram

Request Validation:

The responding server MUST validate the request upon receiving a GetLicensorCertificate request. There are no input parameters, however the **GetLicensorCertificate** request MUST follow the schema specified above in section 2.2.5.1.1.

Response:

A successful **GetLicensorCertificate** response MUST follow the schema specified above in section 2.2.5.1.2. A successful response MUST return the responder's SLC chain. For an unsuccessful request, the server MUST return a fault code. This method throws only Common Fault Codes for the RMS: Server-to-Server Protocol.

Request template:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header>
      <VersionData xmlns="http://microsoft.com/DRM/ServerService">
         <MinimumVersion>string</MinimumVersion>
         <MaximumVersion>string</MaximumVersion>
      </VersionData>
   </soap:Header>
   <soap:Body>
      <GetLicensorCertificate
         xmlns="http://microsoft.com/DRM/ServerService" />
   </soap:Body>
</soap:Envelope>
```

Response template:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header>
```

```
        <VersionData xmlns="http://microsoft.com/DRM/ServerService">
           <MinimumVersion>string</MinimumVersion>
           <MaximumVersion>string</MaximumVersion>
        </VersionData>
     </soap:Header>
     <soap:Body>
        <GetLicensorCertificateResponse
           xmlns="http://microsoft.com/DRM/ServerService">
           <GetLicensorCertificateResult>
              <CertificateChain>
                 <Certificate>xml</Certificate>
                 <Certificate>xml</Certificate>
              </CertificateChain>
           </GetLicensorCertificateResult>
        </GetLicensorCertificateResponse>
     </soap:Body>
  </soap:Envelope>
```

3.4.5 Timer Events

The **Get Licensor Certificate** interface has no timer events.

3.4.6 Other Local Events

The **Get Licensor Certificate** interface has no other local events.

3.5 Group Expansion over SOAP Interface and Group Expansion over Remoting Interface Server Details

An RMS server can be requested to issue a **Use License (UL)** for content that has been published with a policy that specifies a group in a forest that the RMS server cannot contact. In this case, the RMS server needs to contact the appropriate RMS server for that forest and request it to verify group membership. This communication SHOULD use the **Group Expansion over SOAP** interface. This communication MAY use the **Group Expansion over Remoting** interface.<20>

Both interfaces provide one method: **IsPrincipalMemberOf**.

3.5.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document. The conceptual data can be implemented by using a variety of techniques. Any data structure that stores the conceptual data MAY be used in the implementation.

Group membership: A directory or other structure that defines the members of a distribution group.

3.5.2 Timers

The **Group Expansion over SOAP** interface and the **Group Expansion over Remoting** interface have no timers.

3.5.3 Initialization

No specific initialization steps are required for the **Group Expansion over SOAP** interface or the **Group Expansion over Remoting** interface.

3.5.4 Message Processing Events and Sequencing Rules

3.5.4.1 IsPrincipalMemberOf

In the **IsPrincipalMemberOf** operation, the requestor specifies a principal name, the forest of the principal, the target groups for which it needs an answer, and the count of cross-forest calls so far. The responder queries the directory and returns the membership status. IsPrincipalMemberOf MUST return positive if the user is a member of at least one of the specified groups and MUST return false otherwise.

A properly-formed **IsPrincipalMemberOf** request MUST contain valid data for each of these elements.

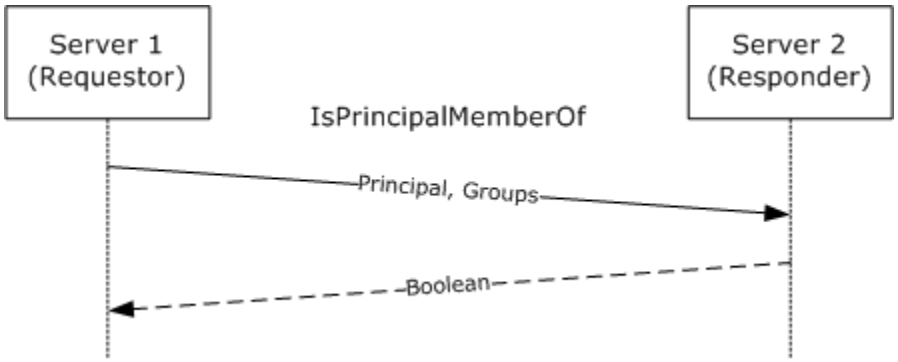

Figure 6: IsPrincipalMemberOf message sequence diagram

Request Validation:

The responding server MUST validate the input parameters upon receiving an IsPrincipalMemberOf request. For the **Group Expansion over SOAP** interface, the IsPrincipalMemberOf request MUST follow the schema specified above in section 2.2.6.1.1.

Data Processing:

For a successful request, the responding server checks the directory for the principal specified in the request and determines whether the principal is a member of one of the groups specified in the request.

If a requested group contains a subgroup in another forest, the responding server SHOULD make a new IsPrincipalMemberOf request to the appropriate server before it responds to the original requestor. In this new request, the count of cross-forest calls so far SHOULD be incremented. This specifies that another server has been involved in servicing this group expansion request. To prevent infinite loops or unacceptably long response times, a server SHOULD reject the request and return a fault if the count exceeds some predefined maximum.

Response:

For the **Group Expansion over SOAP** interface, a successful IsPrincipalMemberOf response MUST follow the schema specified above in section 2.2.6.1.1. For the **Group Expansion over Remoting**

interface, a successful **IsPrincipalMemberOf** response MUST follow the definition specified above in section 2.2.7.1.2.

A successful response MUST return either true or false indicating the group membership status. The status MUST be false if the principal cannot be found or if none of the groups can be found. For an unsuccessful request, the server MUST return a fault code. This method throws only Common Fault Codes for theRMS: Server-to-Server Protocol as specified above in section 2.2.8.

Request template for the Group Membership over SOAP interface:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <VersionData
     xmlns="http://microsoft.com/DRM/GroupExpansionWebService">
      <MinimumVersion>string</MinimumVersion>
      <MaximumVersion>string</MaximumVersion>
    </VersionData>
  </soap:Header>
  <soap:Body>
    <IsPrincipalMemberOf
     xmlns="http://microsoft.com/DRM/GroupExpansionWebService">
      <principalName>string</principalName>
      <principalCrossForest>string</principalCrossForest>
      <targetGroups>
        <string>string</string>
        <string>string</string>
      </targetGroups>
      <crossForestCallsSoFar>int</crossForestCallsSoFar>
    </IsPrincipalMemberOf>
  </soap:Body>
</soap:Envelope>
```

Response template for the Group Membership over SOAP interface:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <VersionData
     xmlns="http://microsoft.com/DRM/GroupExpansionWebService">
      <MinimumVersion>string</MinimumVersion>
      <MaximumVersion>string</MaximumVersion>
    </VersionData>
  </soap:Header>
  <soap:Body>
    <IsPrincipalMemberOfResponse
     xmlns="http://microsoft.com/DRM/GroupExpansionWebService">
      <IsPrincipalMemberOfResult>boolean</IsPrincipalMemberOfResult>
    </IsPrincipalMemberOfResponse>
  </soap:Body>
</soap:Envelope>
```

Definition for the Group Membership over Remoting interface:

```
namespace Microsoft.DigitalRightsManagement.DirectoryServices
{
  interface IRemoteActiveDirectoryServices
  {
    Bool IsPrincipalMemberOf(String principal,
    String principalCrossForest, String[] targetGroups,
    Int32 crossForestCallsSoFar,
    out Microsoft.DigitalRightsManagement.DirectoryServices.Principal
    resultPrincipal);
  }
}
```

3.5.5 Timer Events

The **Group Expansion over SOAP** interface and the **Group Expansion over Remoting** interface have no timer events.

3.5.6 Other Local Events

The **Group Expansion over SOAP** interface and the **Group Expansion over Remoting** interface have no other local events.

4 Protocol Examples

4.1 Accessing Protected Information as a Member of an Authorized Group

An end user requires an authorization token (Use License (UL)) issued by an RMS server to access protected content. One of the conditions evaluated by the server when generating such an authorization token is the user's current membership in any groups that are authorized to consume the information, as specified by the policy of the protected content.

The following example applies to the IsPrincipalMemberOf method for either the **Group Expansion over Remoting** interface or the **Group Expansion over SOAP** interface.

1. Usage policy is extracted from protected content by the application.

   The application extracts (or retrieves) the usage policy (**Publishing License (PL)**) from wherever the application has stored it. Storage of the usage policy associated with protected information (or content) is the responsibility of the application.

2. AcquireLicense method is called.

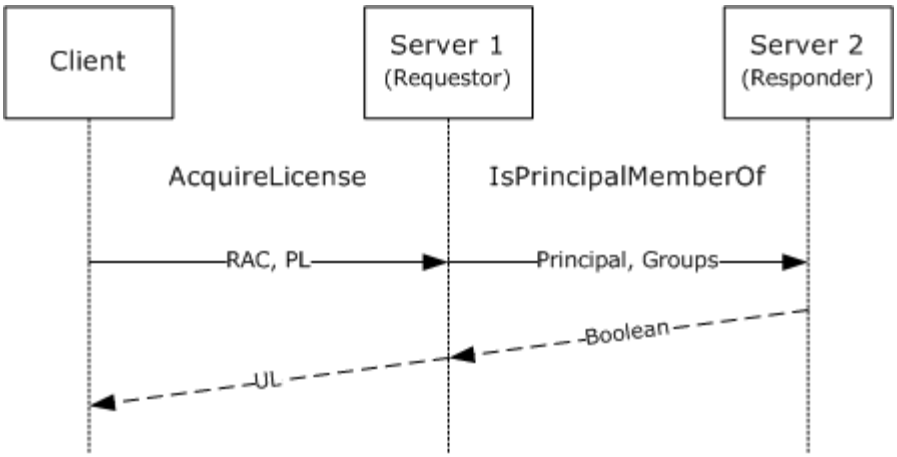

Figure 7: AcquireLicense method is called

The Publishing License (PL) acquired in step 1 represents the complete usage policy issued by the author of the protected content. For the protected content to be consumable by an individual user, an authorization token (Use License (UL)) must be issued by the server. This authorization token expresses what an individual user can do with the protected content.

The client calls the AcquireLicense Web method, providing the requesting user's RMS Account Certificate (RAC) and the protected content Publishing License (PL), and passing any application-specific data provided by the application.

The server verifies that the RMS Account Certificate (RAC) and Publishing License (PL) were issued from an entity or entities it trusts. It then evaluates the full usage policy as it applies to the specific user named in the RAC), including verifying if the user is a member of any groups specified by the author as having authorization to access the protected information (or content).

To verify group membership, the server will either invoke a local **IsPrincipalMemberOf** procedure call to consult the directory in the local **domain**, or it will contact an RMS server located in an external domain to consult that domain's directory via the

IsPrincipalMemberOfremote procedure call (RPC). The result of either is a Boolean response indicating if the named user is a member of a named group (or groups).

The server then issues a Use License (UL) containing a set of usage policy for the user, and signs the Use License (UL). The Use License (UL) is then returned to the client.

3. Decryption of protected information (or content) using client APIs and authorization policy keys occurs.

   The client receiving the Use License (UL) from the server then uses the UL to decrypt and consume the protected content.

4.2 Provisioning an Extranet User

To consume protected content, a user's client machine must contact an RMS server to be provisioned with specific certificates on the machine being used. These certificates can easily be obtained with the proper URLs. Sometimes the necessary URL information is not present on the client machine, either because the machine is not joined to a domain, or it is not joined to the same domain that contains the user's account. As defined in section 1.3.1, the appropriate RMS server for a specific service type can vary depending on the specific user.

To find the relevant RMS server's URL, the client must first contact a general RMS server and invoke the FindServiceLocationsForUser remote procedure call (RPC). The URL for this general RMS server can be found from the policy that has been applied to the protected information (or content). Alternately, the URL for the general server could be found via configuration settings or some application-specific or deployment-specific mechanism.

Once contacted, the general RMS server (server 1 in the diagram below) will in turn contact an appropriate RMS server (server 2 in the diagram below) for the specific user using the FindSeviceLocations method of the **Service Location** interface, and return the appropriate direct URL to the client for subsequent requests to provision the necessary certificates.

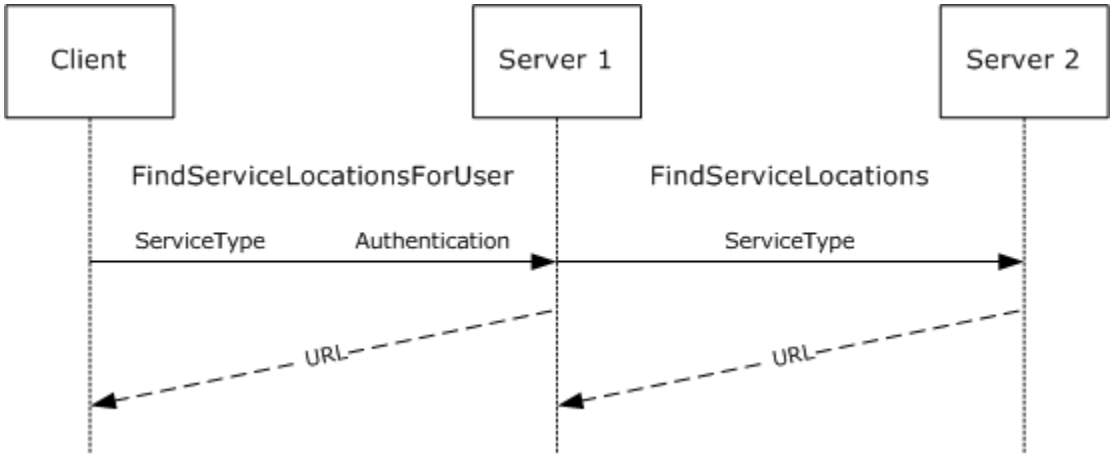

Figure 8: IsPrincipalMemberOf call resulting from AcquireLicense

1. Client identifies general-purpose extranet RMS server URL.

   The client determines the URL of the general-purpose RMS server from the policy applied to the protected content, configuration settings, or an application-specific or deployment-specific mechanism.

2. Client makes a request to FindServiceLocationsForUser remote procedure call (RPC).

   The client makes a request to the FindServiceLocationsForUser Web service, providing the type of service whose URL is being requested and providing authentication of the user's identity.

3. Server 1 makes a FindServerLocations request on behalf of the user.

   If the user authenticates correctly, the server will find the user's home domain in the directory and identify the appropriate RMS server for that domain (server 2). Server 1 will then make a FindServiceLocations request to server 2, specifying the service type that was requested by the client.

4. Server 1 returns results to client.

   Server 1 will then return the results of the request to server 2 to the client.

5. Client contacts server 2 directly, via returned URLs.

   The client will then contact server 2 directly and request the required provisioning certificates.

5 Security

5.1 Security Considerations for Implementers

5.1.1 Service Location Interface

The **Service Location** interface does not communicate any sensitive information. However, it is recommended that communication be done over HTTPS instead of HTTP to mitigate any man-in-the-middle attacks that might allow an attacker to intercept a request for a service location and return the location of an untrustworthy server instead. If an attacker can lure a victim to use this untrustworthy server, protected content published by that victim could be disclosed to the attacker.

Because the **Service Location** interface generally does not require intense processing for the responding server, it is not a severe target for denial of service attacks. However, the impact of such attacks can be further reduced by requiring authentication.

5.1.2 Sub-Enrollment

The **Sub-Enrollment** interface does not communicate any sensitive information. However, it is strongly recommended that access to the **Sub-Enrollment** interface be restricted by requiring authentication and using restrictive access control lists so that the deployment of RMS servers inside a given organization can be controlled and planned. Otherwise, an attacker may be able to deploy a sub-enrolled server that appears legitimate but is actually not sanctioned by the organization. If an attacker can lure a victim to use this untrustworthy server, protected content published by that victim could be disclosed to the attacker. Sub-enrolled servers do not have the capability to license content that has been published against the root RMS server.

Responding to a SubEnroll request results in multiple asymmetric cryptography operations, making it a potential target for denial of service attacks. Restricting access to the interface as described above will reduce this risk.

It is also recommended that communication be done over HTTPS instead of HTTP to mitigate any man-in-the-middle attacks that might allow an attacker to use an untrustworthy server to issue server licensor certificate (SLC) chains to legitimate sub-enrolled servers. While this attack will not result in any information disclosure, it may result in confusion and create administrative burden to correct the situation.

5.1.3 Get Licensor Certificate

The **Get Licensor Certificate** interface does not communicate any sensitive information. However, it is strongly recommended that communication be done over HTTPS instead of HTTP to mitigate any man-in-the-middle attacks. If launched successfully, such an attack could allow an untrustworthy server to be registered as a trusted RMS Account Certificate (RAC) provider for a subordinate RMS server. Properly configured **Secure Sockets Layer (SSL)** is the recommended mitigation.

Because the **Get Licensor Certificate** interface generally does not require intense processing for the responding server, it is not a severe target for denial of service attacks. However, the impact of such attacks can be further reduced by requiring authentication.

5.1.4 Group Expansion

Group Expansion over SOAP and **Group Expansion over Remoting** interfaces could be used to communicate sensitive information. If group membership information is intended to be private for a given organization, it could be disclosed by an attacker using these interfaces. A man-in-the-middle

attack on these interfaces could also result in an unauthorized user gaining access to content by spoofing a positive response to a group expansion request.

Group expansion is also a resource-intensive operation for the directory server(s) being queried, so these interfaces are potential targets for distributed denial of service attacks on directory servers.

Because of this potential exposure, it is very strongly recommended that communication be done over HTTPS instead of HTTP, and that authentication is required, and restrictive access control lists be used to limit access to these interfaces.

5.2 Index of Security Parameters

None.

6 Appendix A: Full WSDL and Interface Definitions

This section contains the full WSDL for each of the SOAP-based interfaces and full definitions for the **Group Expansion over Remoting** interface of the RMS: Server-to-Server Protocol.

6.1 Service Location Interface WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
 xmlns:tns="http://microsoft.com/DRM/ServerService"
 xmlns:s="http://www.w3.org/2001/XMLSchema"
 xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 targetNamespace="http://microsoft.com/DRM/ServerService"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
     targetNamespace="http://microsoft.com/DRM/ServerService">
      <s:element name="VersionData" type="tns:VersionData" />
      <s:complexType name="VersionData">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="MinimumVersion"
           type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="MaximumVersion"
           type="s:string" />
        </s:sequence>
        <s:anyAttribute />
      </s:complexType>
      <s:element name="FindServiceLocations">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="ServiceNames"
             type="tns:ArrayOfServiceLocationRequest" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ArrayOfServiceLocationRequest">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded"
           name="ServiceLocationRequest" nillable="true"
           type="tns:ServiceLocationRequest" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="ServiceLocationRequest">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="Type"
           type="tns:ServiceType" />
        </s:sequence>
      </s:complexType>
      <s:simpleType name="ServiceType">
        <s:restriction base="s:string">
          <s:enumeration value="EnrollmentService" />
          <s:enumeration value="LicensingService" />
          <s:enumeration value="PublishingService" />
```

```
          <s:enumeration value="CertificationService" />
          <s:enumeration value="ActivationService" />
          <s:enumeration value="PrecertificationService" />
          <s:enumeration value="ServerService" />
          <s:enumeration value="DrmRemoteDirectoryServices" />
          <s:enumeration value="GroupExpansionService" />
          <s:enumeration value="LicensingInternalService" />
          <s:enumeration value="CertificationInternalService" />
        </s:restriction>
      </s:simpleType>
      <s:element name="FindServiceLocationsResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
             name="FindServiceLocationsResult"
             type="tns:ArrayOfServiceLocationResponse" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ArrayOfServiceLocationResponse">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded"
            name="ServiceLocationResponse" nillable="true"
            type="tns:ServiceLocationResponse" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="ServiceLocationResponse">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="URL"
           type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="Type"
           type="tns:ServiceType" />
        </s:sequence>
      </s:complexType>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="FindServiceLocationsSoapIn">
    <wsdl:part name="parameters" element="tns:FindServiceLocations"/>
  </wsdl:message>
  <wsdl:message name="FindServiceLocationsSoapOut">
    <wsdl:part name="parameters"
     element="tns:FindServiceLocationsResponse" />
  </wsdl:message>
  <wsdl:message name="FindServiceLocationsVersionData">
    <wsdl:part name="VersionData" element="tns:VersionData" />
  </wsdl:message>
  <wsdl:portType name="ServerSoap">
    <wsdl:operation name="FindServiceLocations">
      <wsdl:input message="tns:FindServiceLocationsSoapIn" />
      <wsdl:output message="tns:FindServiceLocationsSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="ServerSoap" type="tns:ServerSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="FindServiceLocations">
      <soap:operation soapAction=
       "http://microsoft.com/DRM/ServerService/FindServiceLocations"
       style="document" />
```

```
      <wsdl:input>
        <soap:body use="literal" />
        <soap:header message="tns:FindServiceLocationsVersionData"
         part="VersionData" use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
        <soap:header message="tns:FindServiceLocationsVersionData"
         part="VersionData" use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="ServerSoap12" type="tns:ServerSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="FindServiceLocations">
      <soap12:operation soapAction=
       "http://microsoft.com/DRM/ServerService/FindServiceLocations"
       style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
        <soap12:header message="tns:FindServiceLocationsVersionData"
         part="VersionData" use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
        <soap12:header message="tns:FindServiceLocationsVersionData"
         part="VersionData" use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="Server">
    <wsdl:port name="ServerSoap" binding="tns:ServerSoap">
      <soap:address
       location="http://localhost/_wmcs/licensing/server.asmx" />
    </wsdl:port>
    <wsdl:port name="ServerSoap12" binding="tns:ServerSoap12">
      <soap12:address
       location="http://localhost/_wmcs/licensing/server.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

6.2 Sub-Enrollment Interface WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:s1="http://microsoft.com/wsdl/types/"
 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:s="http://www.w3.org/2001/XMLSchema"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:tns="http://microsoft.com/DRM/SubEnrollmentService"
 xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
 targetNamespace="http://microsoft.com/DRM/SubEnrollmentService"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
```

```
targetNamespace="http://microsoft.com/DRM/SubEnrollmentService">
 <s:import namespace="http://microsoft.com/wsdl/types/" />
 <s:element name="SubEnroll">
   <s:complexType>
     <s:sequence>
       <s:element minOccurs="1" maxOccurs="1" name="oInput"
        type="tns:SubEnrollParameters" />
     </s:sequence>
   </s:complexType>
 </s:element>
 <s:complexType name="SubEnrollParameters">
   <s:sequence>
     <s:element minOccurs="1" maxOccurs="1"
      name="CertificatePublicKey"
      type="tns:EnrolleeCertificatePublicKey" />
     <s:element minOccurs="1" maxOccurs="1"
      name="EnrolleeInformation"
      type="tns:EnrolleeServerInformation" />
   </s:sequence>
 </s:complexType>
 <s:complexType name="EnrolleeCertificatePublicKey">
   <s:sequence>
     <s:element minOccurs="0" maxOccurs="1"
      name="aPublicKeyBytes" type="s:base64Binary" />
     <s:element minOccurs="1" maxOccurs="1"
      name="Guid" type="s1:guid" />
   </s:sequence>
 </s:complexType>
 <s:complexType name="EnrolleeServerInformation">
   <s:sequence>
     <s:element minOccurs="0" maxOccurs="1" name="SKU"
      type="s:string" />
     <s:element minOccurs="0" maxOccurs="1" name="Version"
      type="s:string" />
     <s:element minOccurs="0" maxOccurs="1" name="Name"
      type="s:string" />
     <s:element minOccurs="0" maxOccurs="1" name="URL"
      type="s:string" />
   </s:sequence>
 </s:complexType>
 <s:element name="SubEnrollResponse">
   <s:complexType>
     <s:sequence>
       <s:element minOccurs="1" maxOccurs="1"
        name="SubEnrollResult"
        type="tns:SubEnrollResponse" />
     </s:sequence>
   </s:complexType>
 </s:element>
 <s:complexType name="SubEnrollResponse">
   <s:sequence>
     <s:element minOccurs="0" maxOccurs="1"
      name="LicensorCertificateChain" type="tns:ArrayOfString" />
   </s:sequence>
 </s:complexType>
 <s:complexType name="ArrayOfString">
   <s:sequence>
     <s:element minOccurs="0" maxOccurs="unbounded"
```

```
          name="string" nillable="true" type="s:string" />
        </s:sequence>
      </s:complexType>
      <s:element name="VersionData" type="tns:VersionData" />
      <s:complexType name="VersionData">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1"
           name="MinimumVersion" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1"
           name="MaximumVersion" type="s:string" />
        </s:sequence>
      </s:complexType>
    </s:schema>
    <s:schema elementFormDefault="qualified"
     targetNamespace="http://microsoft.com/wsdl/types/">
      <s:simpleType name="guid">
        <s:restriction base="s:string">
          <s:pattern value=
           "[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-
            [0-9a-fA-F]{4}-[0-9a-fA-F]{12}"/>
        </s:restriction>
      </s:simpleType>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="SubEnrollSoapIn">
    <wsdl:part name="parameters" element="tns:SubEnroll" />
  </wsdl:message>
  <wsdl:message name="SubEnrollSoapOut">
    <wsdl:part name="parameters" element="tns:SubEnrollResponse" />
  </wsdl:message>
  <wsdl:message name="SubEnrollVersionData">
    <wsdl:part name="VersionData" element="tns:VersionData" />
  </wsdl:message>
  <wsdl:portType name="SubEnrollServiceSoap">
    <wsdl:operation name="SubEnroll">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
       SubEnrollment Entry Point</documentation>
      <wsdl:input message="tns:SubEnrollSoapIn" />
      <wsdl:output message="tns:SubEnrollSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="SubEnrollServiceSoap"
   type="tns:SubEnrollServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
     style="document" />
    <wsdl:operation name="SubEnroll">
      <soap:operation
       soapAction=
        "http://microsoft.com/DRM/SubEnrollmentService/SubEnroll"
       style="document" />
      <wsdl:input>
        <soap:body use="literal" />
        <soap:header message="tns:SubEnrollVersionData"
         part="VersionData" use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
        <soap:header message="tns:SubEnrollVersionData"
```

```
          part="VersionData" use="literal" />
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="SubEnrollService">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">
    A web service used to enroll additional DRM servers in an enterprise
      </documentation>
      <wsdl:port name="SubEnrollServiceSoap"
       binding="tns:SubEnrollServiceSoap">
        <soap:address location="
         http://rms01/_wmcs/Certification/SubEnrollService.asmx" />
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

6.3 Get Licensor Certificate Interface WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
 xmlns:tns="http://microsoft.com/DRM/ServerService"
 xmlns:s="http://www.w3.org/2001/XMLSchema"
 xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 targetNamespace="http://microsoft.com/DRM/ServerService"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
     targetNamespace="http://microsoft.com/DRM/ServerService">
      <s:element name="GetLicensorCertificate">
        <s:complexType />
      </s:element>
      <s:element name="GetLicensorCertificateResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
             name="GetLicensorCertificateResult"
             type="tns:LicensorCertChain" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="LicensorCertChain">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1"
           name="CertificateChain" type="tns:ArrayOfXmlNode" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="ArrayOfXmlNode">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded"
           name="Certificate" nillable="true">
            <s:complexType mixed="true">
              <s:sequence>
                <s:any />
```

```
            </s:sequence>
          </s:complexType>
        </s:element>
      </s:sequence>
    </s:complexType>
    <s:element name="VersionData" type="tns:VersionData" />
    <s:complexType name="VersionData">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1"
         name="MinimumVersion" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1"
         name="MaximumVersion" type="s:string" />
      </s:sequence>
      <s:anyAttribute />
    </s:complexType>
  </s:schema>
</wsdl:types>
<wsdl:message name="GetLicensorCertificateSoapIn">
  <wsdl:part name="parameters"
   element="tns:GetLicensorCertificate" />
</wsdl:message>
<wsdl:message name="GetLicensorCertificateSoapOut">
  <wsdl:part name="parameters"
   element="tns:GetLicensorCertificateResponse" />
</wsdl:message>
<wsdl:message name="GetLicensorCertificateVersionData">
  <wsdl:part name="VersionData" element="tns:VersionData" />
</wsdl:message>
<wsdl:portType name="ServerSoap">
  <wsdl:operation name="GetLicensorCertificate">
    <wsdl:input message="tns:GetLicensorCertificateSoapIn" />
    <wsdl:output message="tns:GetLicensorCertificateSoapOut" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="ServerSoap" type="tns:ServerSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="GetLicensorCertificate">
    <soap:operation soapAction=
     "http://microsoft.com/DRM/ServerService/GetLicensorCertificate"
      style="document" />
    <wsdl:input>
      <soap:body use="literal" />
      <soap:header message="tns:GetLicensorCertificateVersionData"
       part="VersionData" use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
      <soap:header message="tns:GetLicensorCertificateVersionData"
       part="VersionData" use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="ServerSoap12" type="tns:ServerSoap">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="GetLicensorCertificate">
    <soap12:operation soapAction=
     "http://microsoft.com/DRM/ServerService/GetLicensorCertificate"
     style="document" />
```

```
      <wsdl:input>
        <soap12:body use="literal" />
        <soap12:header message="tns:GetLicensorCertificateVersionData"
         part="VersionData" use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
        <soap12:header message="tns:GetLicensorCertificateVersionData"
         part="VersionData" use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="Server">
    <wsdl:port name="ServerSoap" binding="tns:ServerSoap">
      <soap:address location=
       "http://localhost/_wmcs/licensing/server.asmx" />
    </wsdl:port>
    <wsdl:port name="ServerSoap12" binding="tns:ServerSoap12">
      <soap12:address location=
       "http://localhost/_wmcs/licensing/server.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

6.4 Group Expansion over SOAP Interface WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
 xmlns:tns="http://microsoft.com/DRM/GroupExpansionWebService"
 xmlns:s="http://www.w3.org/2001/XMLSchema"
 xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 targetNamespace="http://microsoft.com/DRM/GroupExpansionWebService"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace=
     "http://microsoft.com/DRM/GroupExpansionWebService">
      <s:element name="IsPrincipalMemberOf">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
             name="principalName" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1"
             name="principalCrossForest" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1"
             name="targetGroups" type="tns:ArrayOfString" />
            <s:element minOccurs="1" maxOccurs="1"
             name="crossForestCallsSoFar" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ArrayOfString">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="string"
           nillable="true" type="s:string" />
        </s:sequence>
      </s:complexType>
```

```
      <s:element name="IsPrincipalMemberOfResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1"
             name="IsPrincipalMemberOfResult" type="s:boolean" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="VersionData" type="tns:VersionData" />
      <s:complexType name="VersionData">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1"
             name="MinimumVersion" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1"
             name="MaximumVersion" type="s:string" />
        </s:sequence>
        <s:anyAttribute />
      </s:complexType>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="IsPrincipalMemberOfSoapIn">
    <wsdl:part name="parameters" element="tns:IsPrincipalMemberOf" />
  </wsdl:message>
  <wsdl:message name="IsPrincipalMemberOfSoapOut">
    <wsdl:part name="parameters"
     element="tns:IsPrincipalMemberOfResponse" />
  </wsdl:message>
  <wsdl:message name="IsPrincipalMemberOfVersionData">
    <wsdl:part name="VersionData" element="tns:VersionData" />
  </wsdl:message>
  <wsdl:portType name="GroupExpansionWebServiceSoap">
    <wsdl:operation name="IsPrincipalMemberOf">
      <wsdl:input message="tns:IsPrincipalMemberOfSoapIn" />
      <wsdl:output message="tns:IsPrincipalMemberOfSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="GroupExpansionWebServiceSoap"
   type="tns:GroupExpansionWebServiceSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="IsPrincipalMemberOf">
      <soap:operation soapAction=
"http://microsoft.com/DRM/GroupExpansionWebService/IsPrincipalMemberOf" style="document"
/>
      <wsdl:input>
        <soap:body use="literal" />
        <soap:header message="tns:IsPrincipalMemberOfVersionData"
         part="VersionData" use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
        <soap:header message="tns:IsPrincipalMemberOfVersionData"
         part="VersionData" use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="GroupExpansionWebServiceSoap12"
   type="tns:GroupExpansionWebServiceSoap">
    <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="IsPrincipalMemberOf">
      <soap12:operation soapAction=
"http://microsoft.com/DRM/GroupExpansionWebService/IsPrincipalMemberOf" style="document"
/>
      <wsdl:input>
        <soap12:body use="literal" />
        <soap12:header message="tns:IsPrincipalMemberOfVersionData"
```

```
          part="VersionData" use="literal" />
        </wsdl:input>
        <wsdl:output>
          <soap12:body use="literal" />
          <soap12:header message="tns:IsPrincipalMemberOfVersionData"
           part="VersionData" use="literal" />
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="GroupExpansionWebService">
      <wsdl:port name="GroupExpansionWebServiceSoap"
       binding="tns:GroupExpansionWebServiceSoap">
        <soap:address location=
         "http://localhost/_wmcs/groupexpansion/groupexpansion.asmx" />
      </wsdl:port>
      <wsdl:port name="GroupExpansionWebServiceSoap12"
       binding="tns:GroupExpansionWebServiceSoap12">
        <soap12:address location=
          "http://localhost/ wmcs/groupexpansion/groupexpansion.asmx" />
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

6.5 Group Expansion over Remoting Interface Full Definitions

```
namespace Microsoft.DigitalRightsManagement.DirectoryServices
{
  interface IRemoteActiveDirectoryServices
  {
    Bool IsPrincipalMemberOf(String principal,
    String principalCrossForest, String[] targetGroups,
    Int32 crossForestCallsSoFar,
    out Microsoft.DigitalRightsManagement.DirectoryServices.Principal
    resultPrincipal);
  }
  enum ExplicitParseEnum
  {
    None = 0,
    PrimaryMail = 1,
     RFC822 = 2
  }
  class Principal
  {
    System.Collections.ListDictionary      _PrincipalIdentifiers;
    System.Collections.Hashtable           _GroupMembership;
    System.Collections.ListDictionary      _ForeignMembers;
    System.Collections.ListDictionary      _parsingDictionary;
    System.Collections.StringCollection    _ContainerObjectGuids;
    String                                 _strObjectGuid;
    String                                 _strOriginationForest;
 Microsoft.DigitalRightsManagement.DirectoryServices.ExplicitParseEnum
                                           _explicitParse;
    Bool                                   _exists;
  }
}
namespace System.Collections
{
  class ListDictionary
  {
```

```
    System.Collections.ListDictionary+DictionaryNode    head;
    Int32                                               count;
    Int32                                               version;
  }
  class ListDictionary+DictionaryNode
  {
    String                                              key;
    System.Object                                       value;
    System.Collections.ListDictionary+DictionaryNode    next;
  }
  class Hashtable
  {
    Int32       LoadFactor;
    Int32       Version;
    Int32       HashSize;
    String[]    Keys;
    Bool[]      Values;
  }
  class StringCollection
  {
    System.Collections.ArrayList    data;
  }
  class ArrayList
  {
    System.Object[]     _items;
    Int32               _size;
    Int32               _version;
  }
}
```

7 Appendix B: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows NT
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista
- Windows Server 2008

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies Windows does not follow the prescription.

<1> Section 1: All existing versions of RMS Server (RMS version 1.0, RMS version 1.0 SP1, RMS version 1.0 SP2, and RMS version 2.0) provide the **Service Location** and **Get Licensor Certificate** interfaces. RMS version 1.0, RMS version1.0 SP1, and RMS version 1.0 SP2 provide the **Sub-Enrollment** interface. RMS version 2.0 does not provide this interface. RMS version 1.0 and RMS version 1.0 SP1 provide the **Group Expansion over Remoting** interface. RMS version 1.0 SP2 provides both the **Group Expansion over Remoting** interface (for backward compatibility) and the **Group Expansion over SOAP** interface. RMS version 2.0 only provides the **Group Expansion over SOAP** interface.

<2> Section 1.3.2: The **Sub-Enrollment** interface is only implemented by RMS version 1.0, RMS version 1.0 SP1, and RMS version 1.0 SP2. Because RMS version 2.0 self-bootstraps, it neither calls nor exposes this interface.

<3> Section 1.3.4: RMS version 1.0 SP2 and RMS version 2.0 implement the **Group Expansion over SOAP** interface.

<4> Section 1.3.5: RMS version 1.0 and RMS version 1.0 SP1 implement the **Group Expansion over Remoting** interface. RMS version 1.0 SP2 implements this interface only for backward compatibility when communicating with RMS version 1.0 or RMS version 1.0 SP1 servers.

<5> Section 1.7: RMS version 1.0 servers, RMS version 1.0 SP1 servers, and RMS version 1.0 SP2 servers run only on Windows Server 2003 (RTM plus any service packs). RMS version 2.0 servers runs only on Windows Server 2008.

<6> Section 2.1: RMS version 1.0 and RMS version 1.0 SP1 do not implement the **Group Expansion over SOAP** interface. RMS version 2.0 does not implement the **Group Expansion over Remoting** interface.

RMS version 2.0 does not implement the **Sub-Enrollment** interface.

RMS version 1.0, RMS version 1.0 SP1, and RMS version 1.0 SP2 support the Simple Object Access Protocol (SOAP) 1.1 (as specified in [SOAP1.1]) only. RMS version 2.0 supports both SOAP 1.1 and SOAP 1.2.

RMS supports HTTPS for securing its communications, although Secure Sockets Layer (SSL) is not configured by default.

<7> Section 2.2.3.1.1: RMS Server requests only one service location at a time.

<8> Section 2.2.3.1.1.3: Not used by RMS 2.0. Only used by RMS 1.0, RMS 1.0 SP1, and RMS 1.0 SP2.

<9> Section 2.2.3.1.1.3: Not used by RMS 1.0 or RMS 1.0 SP1. Only used by RMS 1.0 SP2 and RMS 2.0.

<10> Section 2.2.3.1.1.3: Not used by RMS 1.0, RMS 1.0 SP1, or RMS 1.0 SP2. Only used by RMS 2.0.

<11> Section 2.2.3.1.1.3: Not used by RMS 1.0, RMS 1.0 SP1, or RMS 1.0 SP2. Only used by RMS 2.0.

<12> Section 2.2.4.1.1.2: RMS expects a 1024-bit RSA PKCS#1-encoded public key.

<13> Section 3.1: RMS version 1.0, RMS version 1.0 SP1, RMS version 1.0 SP2, and RMS version 2.0 all use a MinimumVersion and MaximumVersion of "1.0.0.0" for all SOAP requests and responses in the RMS: Server-to-Server Protocol.

<14> Section 3.2.4.1: Not used by RMS 2.0. Only used by RMS 1.0, RMS 1.0 SP1, and RMS 1.0 SP2.

<15> Section 3.2.4.1: Not used by RMS 1.0 or RMS 1.0 SP1. Only used by RMS 1.0 SP2 and RMS 2.0.

<16> Section 3.2.4.1: Not used by RMS 1.0, RMS 1.0 SP1, or RMS 1.0 SP2. Only used by RMS 2.0.

<17> Section 3.2.4.1: Not used by RMS 1.0, RMS 1.0 SP1, or RMS 1.0 SP2. Only used by RMS 2.0.

<18> Section 3.3.1: RMS server currently generates a random 1,024-bit RSA key pair on installation and retains this state.

<19> Section 3.3.4.1: RMS expects a 1024-bit RSA PKCS#1-encoded public key.

<20> Section 3.5: RMS version 1.0 and RMS version 1.0 SP1 use the **Group Expansion over Remoting** interface. RMS version 1.0 SP2 supports both the **Group Expansion over Remoting** interface and the **Group Expansion over SOAP** interface. It will use the **Group Expansion over SOAP** interface whenever possible. RMS version 2.0 only supports the **Group Expansion over SOAP** interface.

8 Index