Interoperability Specification for Backup and Migration Services

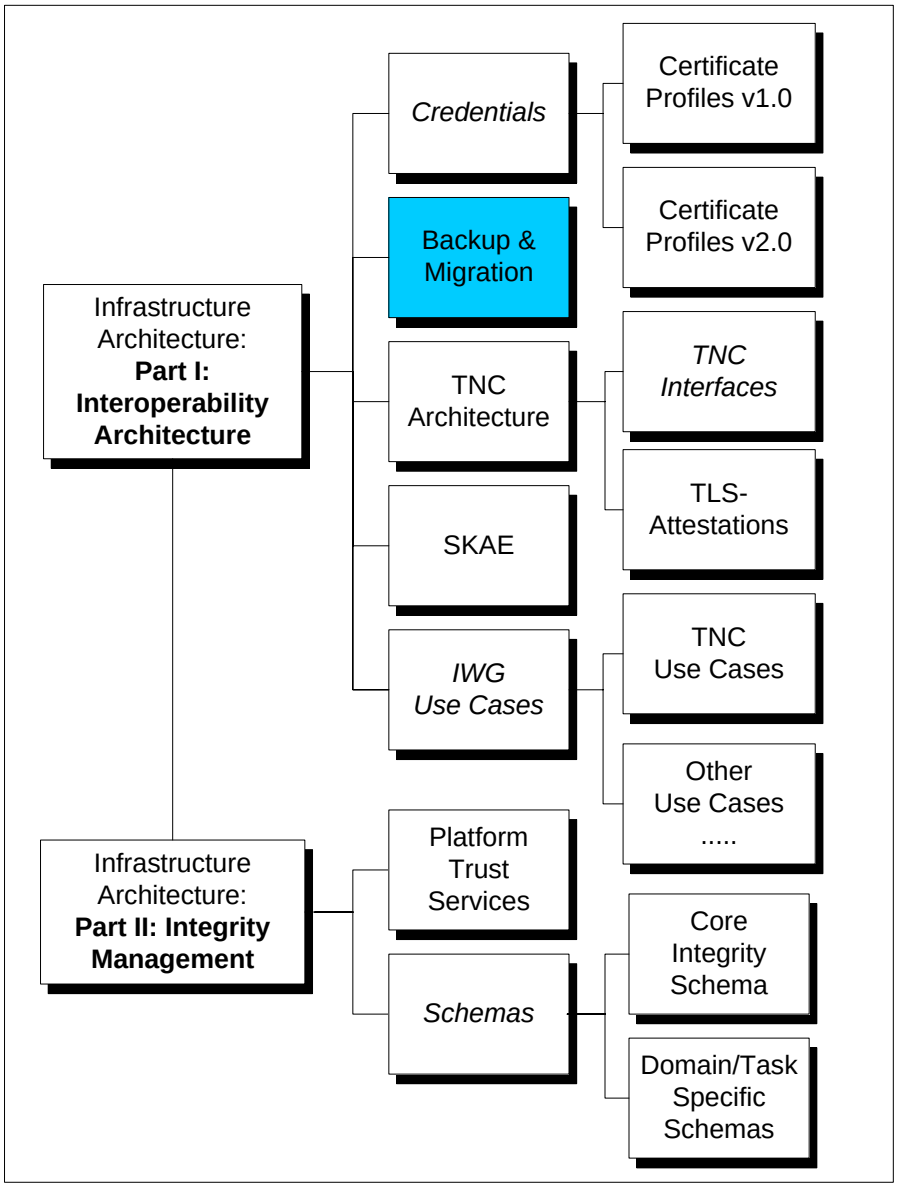
Specification Version: 1.0 Final
Revision 1.0
30 June, 2005
For TPM Family: 1.1b, Level: 1

Contact: techquestions@trustedcomputinggroup.org

TCG PUBLISHED

TCG

IWG Document Roadmap



Revision History

	Document started by Len Veil and Greg Kazmierczak	26 March, 2004
Rev 0.99	Submitted for 60-day internal TCG review	22 December, 2004
Rev 1.0	TCG Board of Directors approval for publication	16 June, 2005

Acknowledgement

The TCG wishes to thank all those who contributed to this specification. This document builds on numerous work done in the various working groups in the TCG.

Special thanks to the members of the IWG contributing to this document:

Boris Balacheff	Hewlett-Packard
Graeme Proudler	Hewlett-Packard
Diana Arroyo	IBM
Steven Bade	IBM
Lee Terrell	IBM
Hubert Braunwarth	Infineon
Markus Gueller	Infineon
Johann Schoetz	Infineon
David Grawrock	Intel
Ned Smith (IWG Co-Chair)	Intel
Monty Wiseman	Intel
Mark Williams	Microsoft
Brad Andersen	SignaCert
Jeff Nisewanger	SUN
Paul Sangster	SUN
Thomas Hardjono (IWG Co-Chair)	Verisign
Greg Kazmierczak (Editor)	Wave Systems
Len Veil (Editor)	Wave Systems
Liping Dai	Wave Systems
Mike Gile	Wave Systems
Tania Lissitskaia	Wave Systems

Table of Contents

1 Introduction

This document is a living product of the Data Migration Task Force of the Infrastructure Workgroup (IWG). It is intended for those interested in understanding or developing interoperable Migration Services for TCG compliant platforms. The detailed specification portions of this document will eventually be incorporated into actual specification documents produced and published by the IWG.

1.1 Document Purpose

This document is intended to serve as the living design document for the development of the Interoperability Specification for Backup and Migration Services of the TCG Infrastructure Workgroup. It details the specification as well as provides a historical record of decisions that were made in developing this specification.

1.2 Scope

The scope of the initial release is limited to specific TPM v1.1b specification relevant Use Cases including TPM v1.2 specification structures used in methods consistent with TPM v1.1b specification. I.e. TPM v1.2 supported scenarios include TPM v1.2 key structures when used in TPM v1.1b migration methods and TPM v1.2 CMKs when the targets of migration are Migration Authorities. TPM v1.2 CMK cases when the targets of migration are keys approved by Migration Selection Authorities are not supported in this specification. Follow-on work will address a more comprehensive set of Use Cases including full support for TPM v1.2 specification dependencies.

1.3 Intended Audience

The intended audience for this document is IT professionals who wish to develop or understand Migration Authorities and Migration Selection Authorities, or who have interest in key Migration operations as they apply to TCG compliant platforms.

Certain portions of this specification may be relevant to professionals who work for computer OEMs and the companies in the OEM supply chain, such as TPM vendors and software vendors, in order to ensure the development of interoperable products.

It is expected that the professionals attempting to comprehend this specification will have a working knowledge of TCG fundaments, especially regarding the TPM and TSS. Additionally, a rudimentary understanding of Internet technologies, web services architecture and distributed computing concepts is highly desirable. Internet messaging technologies include HTTP, SOAP and TLS as well as an understanding of internet data description languages such as XML, XKMS, WSDL and ODRL will be of value to readers. An understanding of Public Key Infrastructure (PKI), certificate encoding technologies, XML Signatures and XML Encryption may also be beneficial.

1.4 Goals

This document is intended to provide:

1. an interoperable reference specification for TCG clients and servers to interact in order to effect Migration services across a private or public network,
2. interoperable services for TCG clients using either MSCAPI or PKCS#11 CSPs,
3. extensibility for other CSPs

In the IWG's effort to promote interoperability, this specification strives to ensure compliance with the WS-I Basic Profile 1.1. This profile defines a set of non-proprietary Web services specifications, along with clarifications, refinements, interpretations and amplifications of those specifications which promote interoperability.

1.5 References to Other Documents

TCG Infrastructure Committee Use Cases, Specification Version 1.0, Revision 0.25, 27 February, 2004, Draft

TCG Specification Architecture Overview, Specification Version 1.1b, Revision 1.2, 28 April, 2004

TCG IWG Building Blocks

TCG Glossary draft, latest version available on-line on the TCG Technical Committee page in the Members Area at www.trustedcomputinggroup.org

TCG Main Specification Version 1.1b, 22 February, 2002

TCG TPM Main Part 2 TPM Structures, Version 1.2 Rev. 79, 10 October, 2004

TCG Software Stack Specification Version 1.10, 20 August, 2003

TCG Software Stack Specification Version 1.2 Differences, Level .21, 5 October, 2004

IETF, XML – Signature Syntax and Processing, W3C Recommendation 12 February, 2002, available at: http://www.w3.org/TR/xmldsig-core/

Web Services Architecture: W3C Working Group Note 11 February, 2004, available at: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/wsa.pdf

Building Interoperable Web Services: WS-I Basic Profile 1.0, Microsoft Web Services Application Note, 2003, available at: http://download.microsoft.com/download/8/d/8/8d828b77-2ab2-4c8e-b389-e23684f12035/WSI-BP.pdf

WS-I Basic Profile Version 1.1, 24 August, 2004, available at: http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

1.6 Definition of Terms

Term	Definition
Migration Blob	A tssBlob, as defined by the TCG standards, which holds the encrypted migratable key which has been migrated out of a TPM. In this document, a tssBlob may be of type 1 (a TCG_Key in support of rewrap migration scheme) or of type 3 (a TCG_Key in support of migrate migration scheme).
Migration Package (MP)	An XML document which contains the parent migratable key and any children keys of the migrating key which a client desires to archive or migrate to a new platform. The MP optionally may contain key specific data as well as application specific data.
Migration Ticket	A tssBlob, owner authorized software token created by the TPM which specifies a public key (usually a Migration Authority public key) which any user can use as the target of a key migration operation. The migration ticket is a mandatory prerequisite to the creation of a migration blob.

Table 1: Definition of Terms

2 Migration Principles

Migration is a TCG specific operation that allows for the secure movement of migratory cryptographic keys from one TCG compliant platform to another compliant platform in such a fashion as to allow the new environment to function in a similar manner, with respect to the usage of the cryptographic keys. These operations are effected with the use of a Migration Authority and a Migration Selection Authority.

A variety of entities may function as one of these Authorities – they may be private or public. As examples, for the enterprise the IT Organization will most commonly be the Authority that the employees use. In the government, it is expected that a particular agency might operate Authorities for their constituents, such as DoD or Department of Homeland Security. In the consumer market it is expected that recognized and trusted service providers will operate Authorities for individuals to subscribe to, potentially Kinkos or Disney or a major financial institution or citizen friendly government agency such as the Post Office or the IRS.

2.1 Key Types

TPMs may generate three types of keys – NMKs, MKs, and CMKs.

2.1.1 Non-Migratable Key (NMK)

A non-mirgratable key is a key which is bound to a single TPM, is (statistically) unique to that TPM, and can not be migrated from that TPM.

2.1.2 Migratable Key (MK)

A key which is not bound to a specific TPM, and with suitable authorization, can be used outside a TPM or moved to another TPM.

2.1.3 Certified Migratable Key (CMK)

Certified Migration Key: a key whose migration from a TPM requires an authorization token created with private keys. The corresponding public keys are incorporated in the CMK and referenced when a TPM produces a credential describing the CMK. If a CMK credential is signed by an AIK, an external entity has evidence that a particular key (1) is protected by a valid TPM and (2) requires permission from a specific authority before it can be copied.

2.2 Migration Entities

2.2.1 Migration Authorities

The Migration Authority (MA) participates in the migration of a migratory key. Migration of a MK or CMK is coordinated by an MA, and the key may be vaulted in the MA or temporally transit through the MA. It is anticipated that public Migration Authorities publish their practices and policies in a Migration Practice Statement (MPS) similar to a Certificate Practice Statement for a Certificate Authority. The MPS must be a human readable description.

2.2.2 Migration Selection Authorities

The Migration Selection Authority (MSA) controls migration of a CMK. In addition to the normal authorizations required to migrate a MK key, the MSA may authorize a CMK migration destination without handling the key itself. The TPM Owner authorizes the migration destination (as usual), and the key owner authorizes the migration transformation (as usual). An MSA authorizes the migration destination as well. MSAs are not supported in this specification, but will be supported in a future version.

2.3 Authorization Values

2.3.1 Random XOR String

A TPM generated random value which is used to provide confidentiality for the encrypted part of the tssBlob from the Migration Authority. This value is output from the TPM, and placed into a tssBlob by the TSS for use by an application or user. The tssBlob enumerated type is not yet defined by TSS.

2.3.2 Usage Authorization

A value (password) known to the user which authorizes the TPM to use a specific private key.

2.3.3 Migration Authorization

A value (password) known to the user which authorizes migration of a specific MK or CMK private key.

2.3.4 Owner Authorization

A value (password) known to the owner which authorizes the TPM to perform a specific action which is available only to the owner of the TPM. Owner authorization may be delegated.

2.3.5 Migration Ticket

Owner authorized software token created by the TPM which specifies a Migration Authority public key which any user can use as the target of a key migration operation. The migration ticket is a mandatory prerequisite to the creation of a Migration Blob. It is placed into a tssBlob by the TSS: Migticket-Blob: enumerated type 6.

2.3.6 CMK Related Structures (tickets)

Reserved for future version of specification addressing TPM v1.2 issues.

2.3.7 Delegation

All usage authorizations may be delegated by the owner or user to other users or trusted processes. Reserved for future versions of the specification addressing TPM v1.2 issues.

3 System Overview

3.1 Migration Package(s)

Migration packages (MPs) are XML documents which are nominally exchanged between TCG clients and Migration Authorities, but may also be directly exchanged between TCG clients using migration re-wrap functionality. A MP contains identifiers for the package, the migrating key, and an optional number of children keys of the subject parent key that is being migrated. For each key which is being migrated, the MP may optionally contain authorization values which control the usage of the key (KeySpecificData) as well as application data (ApplicationSpecificData) which may be required in order to make the key useable in the application context for which it was originally created.

3.2 System Architecture

Figure 1 represents the system architecture as detailed by this specification. Note: this diagram addresses the system architecture for key migration only and it is assumed that all applications and components will communicate directly with the TSS to access other functions.

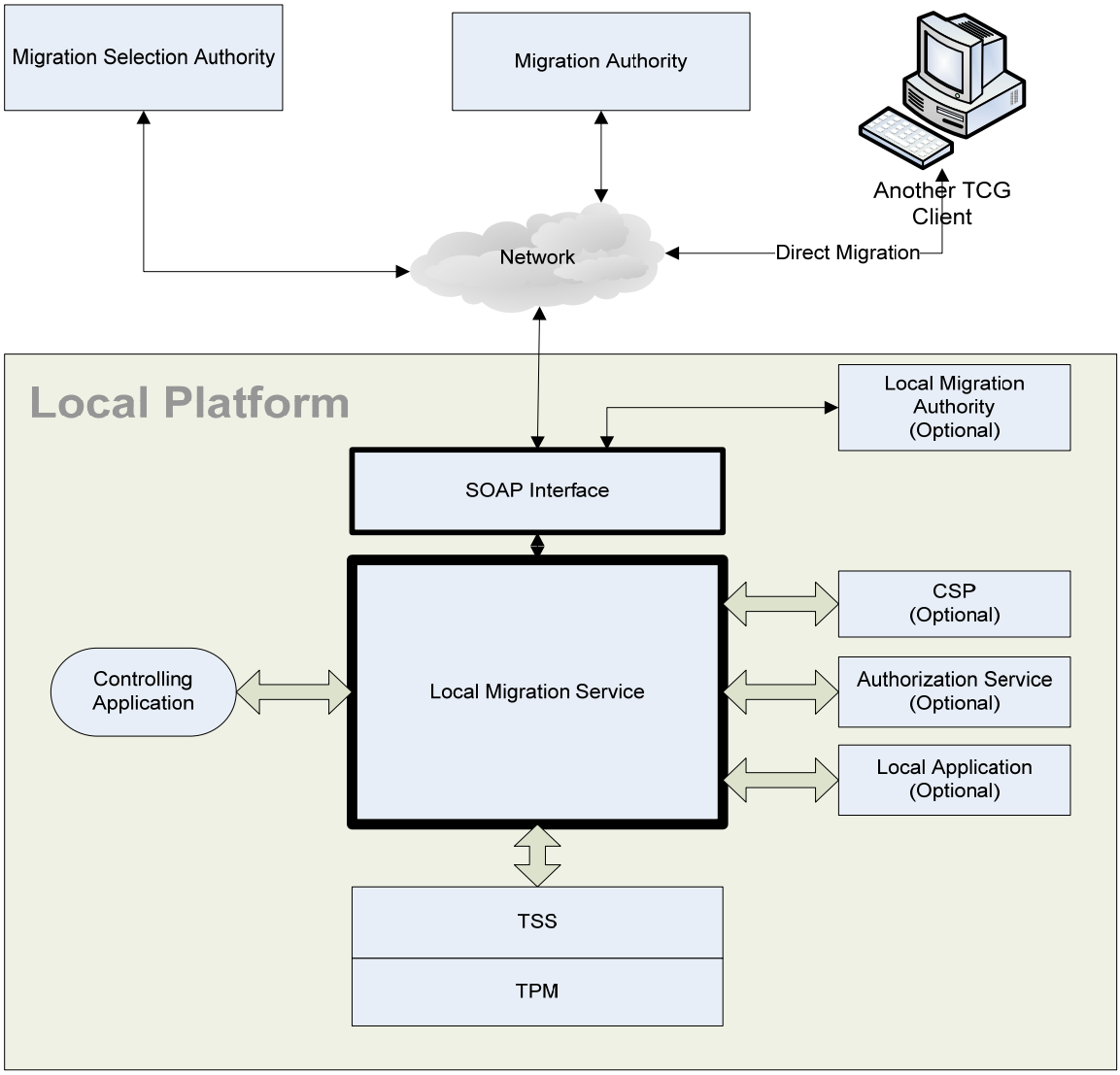


Figure 1: System Diagram

3.2.1 WS-TCG_IWG_Migration Web Service Interface

The Web Service Interface, expressed in WSDL, represents a normative portion of this specification. The WSDL defines the operations required for a compliant TCG client, Migration Authority Web Service, or Migration Selection Authority Web Service to interoperate across heterogeneous platform environments.

3.2.2 Local Migration Service

The Local Migration Service (LMS) is a client service which is responsible for (a) the creation of a Migration Package (MP) which is to be transferred to another entity, either a networked or local Migration Authority or another TCG compliant client, or (b) the parsing of a MP, retrieved either from a remote or local MA or other valid source, which is to be imported into the client. The LMS interacts with the TPM via the TSS as specified in the TCG architecture.

The LMS has the architectural responsibility to identify valid MAs and MSAs, and implement confidentiality and integrity services for MPs.

The LMS may optionally interact with other local services or applications in order to ensure that archived or recovered keys may be successfully restored and used by the intended application(s). LMS interaction with local services and applications may be initiated by the local services and applications or it may be initiated by the LMS – both models are supported. The LMS is platform and implementation dependent, and all LMS functions specified in this document are informative only, suggesting to the implementer items which need to be considered in building a compliant Migration solution.

3.2.3 Controlling Application

The Controlling Application is the local platform application which is responsible for effecting migration operations which are performed under control of the local platform. This controlling application will nominally be responsible for selecting the appropriate MA, determining confidentiality and integrity mechanisms, command the LMS to upload or download MPs from the selected MA, and otherwise interact with the MA for MP administrative operations including querying the MA for valid MPs and requesting the MA to delete MPs.

3.2.4 CSP

The platform CSP may be the repository of specific information which is required for an application to make use of TCG compliant keys in a platform standard fashion. The LMS may be required to interact with the CSP in order that a migrating or restored key has the potential to be used. This interface, being platform specific, is not specified in this standard.

3.2.5 Authorization Service

The Authorization Service may be the repository of authorization information which is required to be presented to the TSS or CSP in order that a key may be used. Authorization Services may also be known as a Password Manager, Password Vault, or other common industry term. The LMS may optionally interact with the Authorization Service in order to collect or restore authorization information which may aid in usability. This interface, being platform specific, is not specified in this standard.

3.2.6 Local Application

The Local Application may be the actual application which is the normal usage environment of a TCG key. The local application could be a file encryption service, a digital signature service, a TLS client, or a myriad of other capabilities. The LMS may optionally interact with the local application in order to ensure that a migrating or restored key has the capacity to be used in a recovered environment. This interface, being platform and application specific, is not specified in this standard.

3.2.7 SOAP Interface

The SOAP interface, or SOAP client, is functionality which may be provided by the client operating environment. It is recognized that some TPMs may inhabit computationally restricted environments, thus this specification may not apply in such instances. Its presence in the system architecture is intended to indicate that the LMS entity will interface with a SOAP toolkit in the client environment by a proprietary fashion, and that all messages sent to or received by a Trusted Platform will nominally be SOAP enveloped web services. The interface represented by the SOAP Interface to Network is the normative WSDL of this specification.

Computationally restricted environments may optionally implement this architecture in split configurations to ease the processing requirements. For example, a TPM-enabled hard disk drive or TPM-enabled peripheral product may choose to implement the LMS on the attached computing platform.

3.2.8 Local Migration Authority

The LMA is intended to indicate that a Migration Authority need not be present on a network, but may run locally on a Trusted Platform. Any Migration Authority, local or remote, need only implement the normative Migration Web Service in order to ensure compatibility with this specification. The connection of the Local Migration Authority to the LMS via SOAP is optional; these 2 components may be implemented to interface directly. It is anticipated that computationally restricted environments may be unable to provide a web services interface for the Local Migration Authority.

3.3 MP Integrity and Confidentiality

As detailed by the system architecture, MPs will nominally be transferred to a Migration Authority over a network. While the architecture is intended to be network independent, certain characteristics may be desirable under specific operating environments, including the use of authenticated channels, encrypted channels, both characteristics, or none.

There are several complex elements within the MP that deserve special consideration. Because the KeySpecificData elements are considered confidential (passwords or equivalent), and the ApplicationSpecificData elements may contain PII, a mechanism must be provided to allow these items to be protected during transmission, protected from the Migration Authority, or protected on a local client. Additionally, to provide for integrity of the MP, a client signature is also supported.

While this specification details an HTTP/SOAP interface, consideration has been provided for the transport of MPs over (a) a standard HTTP/SOAP interface, (b) an interface using WS-Security capabilities, (c) a TLS channel, and (d) a proprietary transfer mechanism (including file copy). Specific elements in the MP, as well as the MP itself, may be protected in a variety of manners depending on the transport.

3.3.1 Confidentiality

MPs are nominally transferred to a Migration Authority under one of two usage scenarios. In the enterprise, the primary purpose of placing a MP in a corporate server may be to ensure that the data which those keys protect are recoverable in the event that an employee is no longer able to provide access to their information or that the platform is no longer available. In the consumer domain, the primary purpose of creating an MP is to allow the user to recover information in the event that a platform is lost or if the user wishes to upgrade to a new platform. While the functionality is essentially the same, the need to provide confidentiality of the data is different.

Enterprises want to ensure that only authorized users have the ability to view or recover authorization values and PII, while the consumers wish to ensure that only they or users whom they have specifically authorized are able to view or recover their PII. Thus, the necessity to protect and recover information within an MP is role based.

Confidentiality of the MP elements is provided for by symmetric encryption. The symmetric key is then protected by the public key of the authority that is being entrusted to hold or process the data elements. The trust properties of the public keys which are used to protect the symmetric encryption key are outside the scope of this specification.

The credentials used to ensure the confidentiality of the MP may be dependent on the needs of the migration services. Table 2 lists the options:

Encrypting Key	Issued to:	Description	Confidentiality
MA Pub Key	MA	Same key that Is used to encrypt the migrating key	MP confidential data available to MA
Platform Secret	Platform	Local key unknown to MA. Key may be transferred to another platform out of band using TCG migration methods.	MP confidential data may only be decrypted by a platform (upon a restore) that possesses encrypting key
User Secret	User	Local user key unknown to MA – key may be portable and transferred out of band (e.g. via use of a hardware token) or via TCG migration methods.	MP confidential data may only be decrypted by a user (upon a restore) that possesses the encrypting key.
Random XOR Key	User	Output of creating the MP. Random XOR Key is NOT transferred to MA in MP.	MP clear text private key and / or successful migration to target TPM may only be recovered by recipient of Random XOR key. If not transferred to MA, then MA can unwrap and rewrap for target, but can not access clear text private key.
None	N/A	N/A	MP confidential data may be viewed by MA or any subsequent recipient

Table 2: Migration Package Confidentiality Options

3.3.2 Integrity

When MPs are resident on a Migration Authority or in transit, they should be protected from tampering. While this may be accomplished through a variety of mechanisms, the preferred approach within this standard is to provide a digital signature over the MP. The signature, if used, should be performed using a trusted key owned by the creator of the MP. The trust properties of the signing key are outside the scope of this specification, however it is suggested that the signing key be of equivalent strength or stronger than the migrating key.

The credentials used to ensure the integrity of the MP may be dependent on the needs of the migration services. Table 3 lists the options:

Credential Type	Issued to:	Description	Integrity Verification Check
User Public	User	Identity credential known to MA	MP integrity can be verified by MA or user
User Secret	User	Identity credential not known to MA	MP integrity can only be verified by user
Platform Public	Platform	Platform identity credential known to MA	MP integrity can be verified by MA or platform owner
Platform Secret	Platform	Platform identity credential not known to MA	MP integrity can only be verified by platform owner
User/Platform Public	User	User identity credential bound to platform and known to MA, can only be used on that particular platform	MP integrity can be verified by MA or platform owner
User/Platform Secret	User	User identity credential bound to platform and not known to MA, can only be used on that particular platform	MP integrity can be verified only by platform owner
tpmCerityInfo	TPM	Put tcgData (tpmCertifyInfo) as extension to ds:KeyInfo and upload with MP	
None	N/a	User created key pair held by user	MP integrity can only be verified by user

Table 3: Migration Package Verification Options

The use of certified credentials, credentials backed by AIKs, is highly recommended.

3.3.3 Platform Authenticity

Prior to accepting requests from a client, a MA may have a responsibility to authenticate the user and / or platform making the request for services. If the interaction between a client and a MA is not session-based (e.g. via a token provided by the MA), then the MA may need to re-authenticate prior to each request for security and privacy purposes. It is expected that future versions of this specification will provide a mechanism to authenticate a platform and user – perhaps based on SAML tokens. Prior to the future version of this specification, WS-SecureConversation, TLS-Att (TLS with attestation extension currently under development in the IWG), and SAML are all good alternatives.

It is expected that one or more of the many currently existing user authentication mechanisms can be layered on top of the WSDL detailed in this specification and is out of scope for this document.

When migrating CMKs, the MA may have sufficient information to authenticate the platform from the AIK used to certify the CMK (for an upload) or the parent of the CMK (for a download). However, if a session-based connection is not maintained, this does not provide sufficient platform authentication for the other functions offered by the MA. When migrating a MK, an AIK may not be used to certify the MK

and thus this information is not available for the upload. However, appropriate information may be available for the download (certify the new parent of the migrating key is a TPM keys with an AIK).

If an authenticating protocol such as TLS-Attestation is used to connect and secure the communication with the MA, then the MA maintains authentication information for the contents of the session.

Authenticating the MA is not addressed in this document other than by the MA's certificate and optionally by attestation of the MA's key. If the MA's key is provided to the platform in an out of band manner, it is the platform owner's responsibility to determine the MA's authenticity.

3.4 MP Security

MPs may be sent using a variety of RPC transport protocols. Depending on the mechanism used, the confidentiality and integrity implementations may vary within the defined MP schema.
Infrastructures which transport migration packages over public or private networks should comprehend the inherent security capabilities of the transport mechanism and take appropriate actions to protect sensitive data.

The following sections identify likely environments and possible scenarios which should be addressed by the service infrastructure.

3.4.1 No Security

When transporting using a protocol which has no inherent security. The signature over the MP is RECOMMENDED in order to ensure integrity of the data during transmission.

If the user wishes to ensure confidentiality of the KeySpecificData and ApplicationSpecificData, the encKeySpecificData and encApplicationSpecificData should be encrypted using a key which is held by the user. If no confidentiality from the MA is required, these elements should be encrypted using the MA public key to protect the symmetric key.

Plaintext transmission of the MP without both integrity and confidentiality is highly discouraged.

3.4.2 SSL

For MPs which are transmitted over an SSL session, the channel is protected providing confidentiality during transmission. The recommended approach is to use both server and client authentication to verify that a proper TCG client is communicating with a valid MA. If the user wishes to ensure confidentiality of the KeySpecificData and ApplicationSpecificData, the encKeySpecificData and encApplicationSpecificData should be encrypted using a key which is held by the user. If no confidentiality from the MA is required, these elements may be assembled with no encryption. Integrity of the data is enforced by the end-to-end nature of the SSL channel.

3.4.3 WS-Security

When using WS-Security, the same properties apply as for SSL. Both client and MA authentication should be used, and the confidentiality of the user data will vary based on the requirements to recover data by the MA or to protect data from disclosure to the MA. Using WS-Security does not imply anything about what to sign or encrypt. A decision on which components of the message need to be signed and encrypted and what keys should be used should be determined by 3.3.1 and 3.3.2.

4 Migration Web Service

4.1 Historical Perspective on Selection of Web Services

Web Services (WS) was determined to be the proper approach to the development of the standard for clients to interact with Migration services. This document's strategy is to focus on aspects of Services Oriented Model (SOM), and then to enhance later with Policy.

Rejected were DCOM and CORBA, UPnP, and ebXML

4.2 Migration Package(s)

This section details the schema and element definitions of MPs.

4.2.1 MP Schema

The Migration Package Schema is expressed in XML using XML Schema. This section provides the schema detail and describes the various elements of the schema

The following listing is the schema represented in text form. This file containing the executable schema representation, MigrationPackage_1-00_1-00.xsd, may be located at:
http://www.trustedcomputinggroup.org/XML/SCHEMA

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.trustedcomputinggroup.org/XML/SCHEMA/MigrationPackage_1-00_1-00"
xmlns:mp="http://www.trustedcomputinggroup.org/XML/SCHEMA/MigrationPackage_1-00_1-00"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
attributeFormDefault="unqualified"
version="0.01">
    <import namespace="http://www.w3.org/2001/04/xmlenc#" schemaLocation="http://www.w3.org/TR/2002/REC-
xmlenc-core-20021210/xenc-schema.xsd"/>
    <complexType name="KeyDescriptor">
        <sequence>
            <element name="label" type="string" nillable="false"/>
            <element name="UUID" type="string" nillable="false"/>
            <element name="parentUUID" type="string" nillable="false"/>
            <element name="description" type="string" default="Key" nillable="true"/>
        </sequence>
    </complexType>
    <complexType name="ChildKey">
        <sequence>
            <element name="keyName" type="mp:KeyDescriptor" nillable="false"/>
            <element name="tsskey-Blob" type="base64Binary" nillable="false"/>
            <element name="encKeySpecificData" type="xenc:EncryptedDataType" minOccurs="0"/>
            <element name="encKeySpecDataKey" type="xenc:EncryptedKeyType" minOccurs="0"/>
            <element name="encApplicationSpecificData" minOccurs="0" maxOccurs="unbounded">
                <complexType>
                    <complexContent>
                        <extension base="xenc:EncryptedDataType"/>
                    </complexContent>
                </complexType>
            </element>
            <element name="encApplicationSpecificDataKey" type="xenc:EncryptedKeyType" minOccurs="0"/>
        </sequence>
        <attribute name="storageType" use="optional" default="System">
            <simpleType>
                <restriction base="string">
                    <enumeration value="User"/>
                    <enumeration value="System"/>
```

```
            </restriction>
        </simpleType>
    </attribute>
    <attribute name="storageLocation" type="string" use="optional" default="TSS"/>
    <attribute name="itemID" type="ID" use="optional"/>
</complexType>
<complexType name="MigratingKey">
    <sequence>
        <element name="keyName" type="mp:KeyDescriptor" nillable="false"/>
        <element name="tssBlob" type="mp:TssBlobType" nillable="false"/>
        <element name="encKeySpecificData" type="xenc:EncryptedDataType" minOccurs="0"/>
        <element name="encKeySpecDataKey" type="xenc:EncryptedKeyType" minOccurs="0"/>
        <element name="encApplicationSpecificData" minOccurs="0" maxOccurs="unbounded">
            <complexType>
                <complexContent>
                    <extension base="xenc:EncryptedDataType"/>
                </complexContent>
            </complexType>
        </element>
        <element name="encApplicationSpecificDataKey" type="xenc:EncryptedKeyType" minOccurs="0"/>
    </sequence>
    <attribute name="storageType" type="string" use="optional" default="System"/>
    <attribute name="storageLocation" type="string" use="optional" default="TSS"/>
    <attribute name="itemID" type="ID" use="optional"/>
</complexType>
<complexType name="ItemType"/>
<complexType name="PlatformInfoType">
    <sequence>
        <element name="TPMInfo" type="mp:InstanceInfoType"/>
        <element name="TSSInfo">
            <complexType>
                <sequence>
                    <element name="TCS">
                        <complexType>
                            <sequence>
                                <element name="TCSInfo" type="mp:InstanceInfoType"/>
                                <element name="TCSPlatformType" type="string"/>
                            </sequence>
                        </complexType>
                    </element>
                    <element name="TSP" type="mp:InstanceInfoType"/>
                </sequence>
            </complexType>
        </element>
    </sequence>
</complexType>
<complexType name="MigrationPackageType">
    <complexContent>
        <extension base="mp:ItemType">
            <sequence>
                <element name="packageInfo">
                    <annotation>
                        <documentation>Information identifying Migration Package</documentation>
                    </annotation>
                    <complexType>
                        <sequence>
                            <element name="itemID" type="ID"/>
                            <element name="label" type="string" nillable="true"/>
                            <element name="created" type="dateTime" nillable="true"/>
                            <element name="modified" type="dateTime" nillable="true"/>
                            <element name="description" type="string" default="Migration Package" nillable="true"/>
                            <element name="version" minOccurs="0">
                                <complexType>
                                    <simpleContent>
                                        <extension base="string">
                                            <anyAttribute namespace="##other"/>
```

```
                                        </extension>
                                    </simpleContent>
                                </complexType>
                            </element>
                        </sequence>
                        <attribute name="Id" type="ID" use="optional"/>
                    </complexType>
                </element>
                <element name="mpHarvester" type="mp:InstanceInfoType" minOccurs="0">
                    <annotation>
                        <documentation>Harvester identifies the component (tool) that was used to collect Migration
                          Package data. This is the LMS in the architecture diagram.</documentation>
                    </annotation>
                </element>
                <element name="platformInfo" nillable="false" minOccurs="0">
                    <annotation>
                        <documentation>Identifies TPM and TSS versions used to collect key data</documentation>
                    </annotation>
                    <complexType>
                        <complexContent>
                            <extension base="mp:PlatformInfoType">
                                <attribute name="Id" type="ID" use="optional"/>
                            </extension>
                        </complexContent>
                    </complexType>
                </element>
                <element name="key" nillable="false">
                    <annotation>
                        <documentation>Migrating Key</documentation>
                    </annotation>
                    <complexType>
                        <complexContent>
                            <extension base="mp:MigratingKey"/>
                        </complexContent>
                    </complexType>
                </element>
                <element name="childKey" minOccurs="0" maxOccurs="unbounded">
                    <annotation>
                        <documentation>Child(ren) of Migrating Key</documentation>
                    </annotation>
                    <complexType>
                        <complexContent>
                            <extension base="mp:ChildKey"/>
                        </complexContent>
                    </complexType>
                </element>
                <element name="migrationPackageSignature" type="ds:SignatureType" minOccurs="0"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<complexType name="ApplicationSpecificDataType">
    <sequence>
        <element name="serviceID" type="string" minOccurs="0"/>
        <element name="certificate" type="ds:KeyInfoType" minOccurs="0" maxOccurs="unbounded"/>
        <element name="cspContainerInfo" minOccurs="0">
            <complexType>
                <sequence>
                    <element name="cspType">
                        <complexType>
                            <choice>
                                <element name="msCAPI" type="string" fixed="msCAPI"/>
                                <element name="pkcs11" type="string" fixed="pkcs11"/>
                                <element name="cdsa" type="string" fixed="cdsa"/>
                            </choice>
                        </complexType>
```

```
                    </element>
                    <element name="cspContainerData" type="base64Binary"/>
                </sequence>
            </complexType>
        </element>
        <element name="policy" type="base64Binary" minOccurs="0" maxOccurs="unbounded"/>
        <any namespace="##other" processContents="lax" minOccurs="0"/>
    </sequence>
</complexType>
<complexType name="KeySpecificDataType">
    <sequence>
        <element name="serviceID" type="string" nillable="true" minOccurs="0"/>
        <element name="xorKey" type="base64Binary" minOccurs="0"/>
        <element name="keyUsageAuth" type="base64Binary" minOccurs="0"/>
        <element name="keyMigrationAuth" type="base64Binary" minOccurs="0"/>
        <element name="attestationInfo" type="mp:CertifyInfoType" minOccurs="0"/>
        <any namespace="##other" processContents="lax" minOccurs="0"/>
    </sequence>
</complexType>
<element name="MigrationPackage">
    <annotation>
        <documentation>TCG Migration Package</documentation>
    </annotation>
    <complexType>
        <complexContent>
            <extension base="mp:MigrationPackageType"/>
        </complexContent>
    </complexType>
</element>
<complexType name="CertifyInfoType">
    <sequence>
        <element name="certifyInfo" type="base64Binary"/>
        <element name="certifyInfoSignature" type="ds:SignatureType"/>
    </sequence>
</complexType>
<complexType name="TssBlobType">
    <choice>
        <element name="migKey-Blob" nillable="false">
            <complexType>
                <simpleContent>
                    <extension base="base64Binary">
                        <attribute name="scheme" type="string" fixed="migrate"/>
                    </extension>
                </simpleContent>
            </complexType>
        </element>
        <element name="key-Blob" nillable="false">
            <complexType>
                <simpleContent>
                    <extension base="base64Binary">
                        <attribute name="scheme" type="string" fixed="rewrap"/>
                    </extension>
                </simpleContent>
            </complexType>
        </element>
    </choice>
</complexType>
<complexType name="InstanceInfoType">
    <sequence>
        <element name="Vendor">
            <complexType>
                <sequence>
                    <element name="VendorName" type="string"/>
                    <element name="VendorID" type="integer" minOccurs="0"/>
                </sequence>
            </complexType>
```

```
            </element>
            <element name="Model" minOccurs="0">
                <complexType>
                    <sequence>
                        <element name="Name" type="string">
                            <annotation>
                                <documentation>Marketing Name</documentation>
                            </annotation>
                        </element>
                        <element name="Number" type="string" minOccurs="0">
                            <annotation>
                                <documentation>Model Number</documentation>
                            </annotation>
                        </element>
                        <element name="SerialNum" type="string" minOccurs="0">
                            <annotation>
                                <documentation>Serial Number</documentation>
                            </annotation>
                        </element>
                        <element name="MachineType" type="string" minOccurs="0">
                            <annotation>
                                <documentation>Vendor specific platform classification</documentation>
                            </annotation>
                        </element>
                    </sequence>
                </complexType>
            </element>
            <element name="Version">
                <complexType>
                    <sequence>
                        <element name="SpecRevMajor" type="integer"/>
                        <element name="SpecRevMinor" type="integer"/>
                        <element name="SpecRevLetter" type="string" minOccurs="0"/>
                        <element name="VendorRevMajor" type="integer" nillable="true"/>
                        <element name="VendorRevMinor" type="integer" nillable="true"/>
                    </sequence>
                </complexType>
            </element>
            <element name="MfgDate" type="date" minOccurs="0"/>
            <element name="PatchLevel" type="string" minOccurs="0"/>
            <element name="VendorSpecific" type="string" minOccurs="0"/>
        </sequence>
        <attribute name="Id" type="ID" use="optional"/>
    </complexType>
</schema>
```

Figure 2, Figure 3, and Figure 4 represent the same information in diagram form. Signatures, certificates, and keys have not been expanded in diagrams as they have been instantiated in standard W3C XML Signature Syntax (xml-dsig) and XML Encryption Syntax (xml-enc) formats.

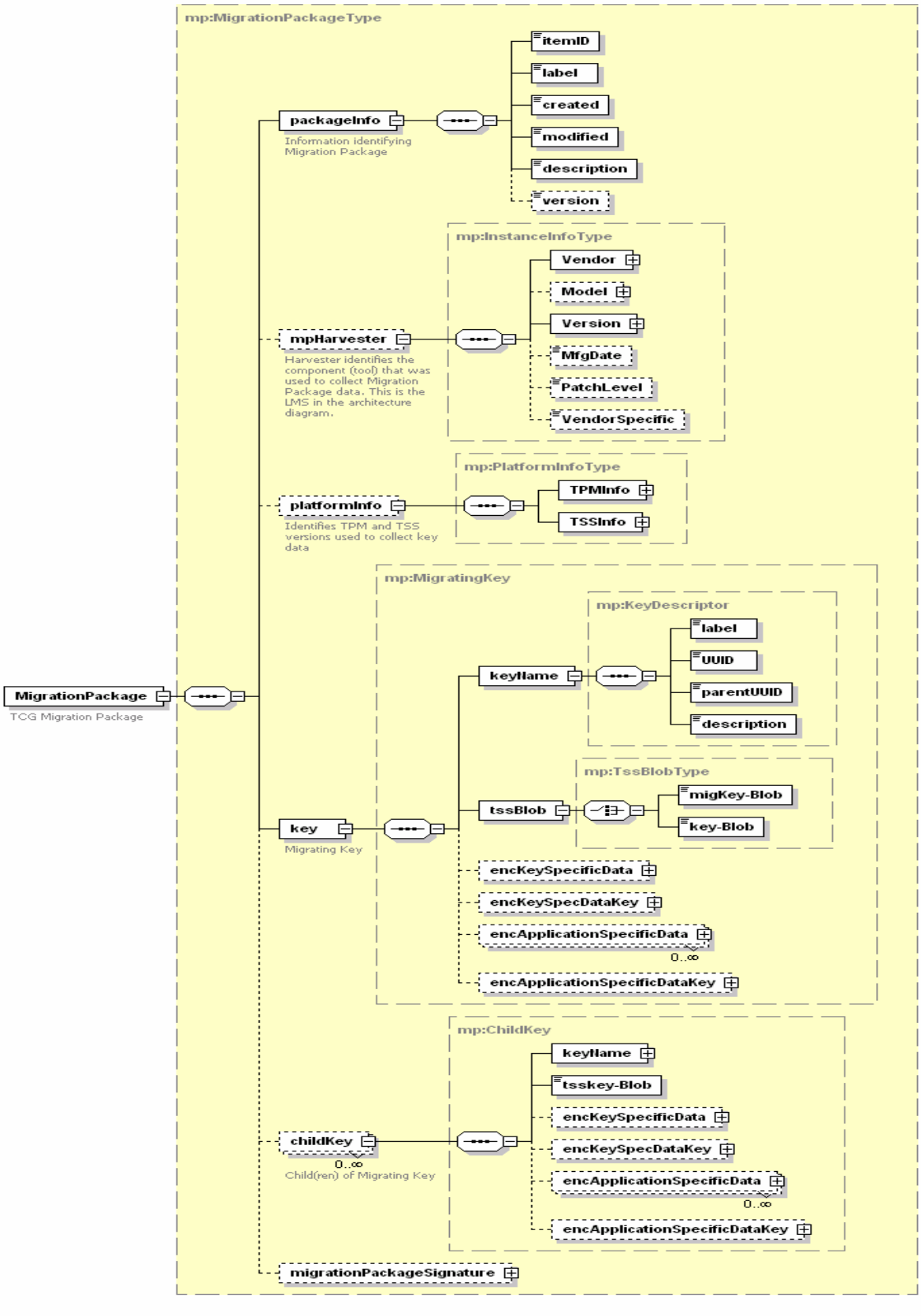


Figure 2: Migration Package Schema

While the Migration Package schema might at first glance appear overly complex and burdensome, a minimal migration package may be constructed containing only the MP Information (ItemID, Label, Created Date, Modified Date, and Description), the Key Name, and the Migrating Key itself (tssBlob). All other elements of the schema are optional. As long as a platform has the ability to create an XML document, even computationally constrained platform will be able to create migration packages.

4.2.1.1 MigrationPackageType

The Migration Package allows for the following elements:

Element	Type	Required	Reference
PackageInfo:ItemID	ID	Y	Unique identifier which the LMS must assign to a migration package
PackageInfo:Label	String [0..255]	Y	Text label to describe migration package, created by LMS
PackageInfo:Created	DateTime	Y	Time of initial creation of MP
PackageInfo:Modified	DateTime	Y	Time of last modification of MP
PackageInfo:Description	String	Y	Package description which may provide additional information to the consumer of the migration package.
PackageInfo:Version	String	N	Package Version or Type – for vendor differentiation and extensibility
PlatformInfo	PlatformInfo	N	TPM and TSS (Both TCS and TSP) Vendor and Version Information
MPHarverstor	InstanceInfo	N	LMS Information
Key	MigratingKey	Y	Subject key which is being migrated
ChildKey	ChildKey	N	One or more children key of subject key which is being migrated
migrationPackageSignature	ds:SignatureType	N	Integrity signature over entire Migration Package

Table 4 – MigrationPackageType

The Harvester is the component which collects data to insert in a MP, constructs the MP XML document, and sends the MP to its destination. In addition, the Harvester receives MPs and comprehends how to retrieve data from the MP. It is assumed that entire Migration Packages can be uploaded to a MA and downloaded from a MA. In Figure 1, the LMS fulfills the Harvester function. If a user desires to download a child key from a Migration Package and not the entire Migration Package, a Migration Authority may not be able to service the request. In this case, a MA will sufficient data to decrypt / unwrap the migrating key to access it's private key and use the private key to create a

migration blob for the child key. The MA may not be able to decrypt the Migration Package if either the MP is encrypted with a key the MA can not access or the key's XOR string is not provided, or if the MA does not have the cryptographic capabilities beyond those normally available in a MA.

4.2.1.2 InstanceInfoType

The MP Harvester, TPM, and TSS information is captured in the InstanceInfo Type. The Harvester is the component that collects data to insert in a MP and is equivalent to

Element	Type	Required	Reference
Vendor:VendorName	String	Y	Vendor Name
Vendor:VendorID	Integer	N	Vendor ID
Model:Name	String	Y	Model Name
Model:Number	String	N	Model Number
Model:SerialNum	String	N	Model Serial Number
Model:MachineType	String	N	Model Machine Type
Version:SpecRevMajor	Integer	Y	TCG Spec Rev Major
Version:SpecRevMinor	Integer	Y	TCG Spec Rev Minor
Version:SpecRevLetter	String	N	TCG Spec Rev Letter
Version:RevMajor	Integer	Y	Vendor Rev Major
Version:RevMinor	Integer	Y	Vendor Rev Minor
MPHarverstor:MfgDate	Date	N	Component Manufacture Date
MPHarverstor:PatchLevel	String	N	Component Patch Level
MPHarverstor:VendorSpecific	String	N	Component Vendor Specific Data

Table 5 - InstanceInfoType

4.2.1.3 KeyDescriptorType

Within both the Migrating Key and any optional Child Keys is the keyName element whose contents describe the identifying properties of the key. Properties such as key size (in bits) are contained in the tssBlob element. The KeyDescriptorType provides for the following elements:

Element	Type	Required	Reference
label	String	Y	Text label associated with key which may be meaningful to user
UUID	String	Y	*From the TSS_KM_KEYINFO struct (TSS 1.1, section 2.5.5)* The UUID of the key registered in the persistent storage of the TSS Key Manager.
parentUUID	String	Y	*From the TSS_KM_KEYINFO struct (TSS 1.1, section 2.5.5)* The UUID the parent key which wraps the key addressed by *keyUUID* is registered in the persistent storage of the TSS Key Manger.
description	String	Y	Description that may provide additional information to the consumer of the key package.

Table 6 - KeyDescriptorType

4.2.1.4 tssBlob

The TssBlob contains a full TCG_Key structure. If Migration Scheme is "migrate", then the encData field of TCG_Key is a MIGRATE_ASYMKEY structure and the tssBlob is of enumerated type 3: MigKey-Blob. If the Migration Scheme is "rewrap", then the encData field of TCG_Key is a STORE_ASYMKEY structure (this is what a normal key structure contains when resident on a platform) and the tssBlob is of enumerated type 1: Key-Blob. In both cases, the contents of the TCG_Key structure will detail key properties.

For the children keys, the TCG_Key should be populated by a TssBlob of type Key-Blob. In this case, TssBlob contains a TCG_Key structure where by definition the encData field is a STORE_ASYMKEY structure.

See the Portable Data section of the TSS 1.1 Specification for more detail.

4.2.1.5 EncKeySpecificDataType & encKeySpecificDataKey

The encKeySpecificDataType is the xml-enc encrypted KeySpecificData. The encKeySpecificDataKey is the xml-enc encrypted symmetric key which is used to protect the KeySpecificData.

Element	Type	Required	Reference
encKeySpecificDataType	xenc:EncryptedDataType	N	Xml-enc encrypted KeySpecificData.
encKeySpecificDataKey	xenc:EncryptedKeyType	N	Xml-enc encrypted symmetric key which is used to protect KeySpecificData.

Table 7

4.2.1.6 KeySpecificDataType

Key authorization values, XOR Key, and attestation information is optionally available in the KeySpecificData field. This structure also contains an element extension for unanticipated key specific data.The KeySpecificDataType has the following schema:

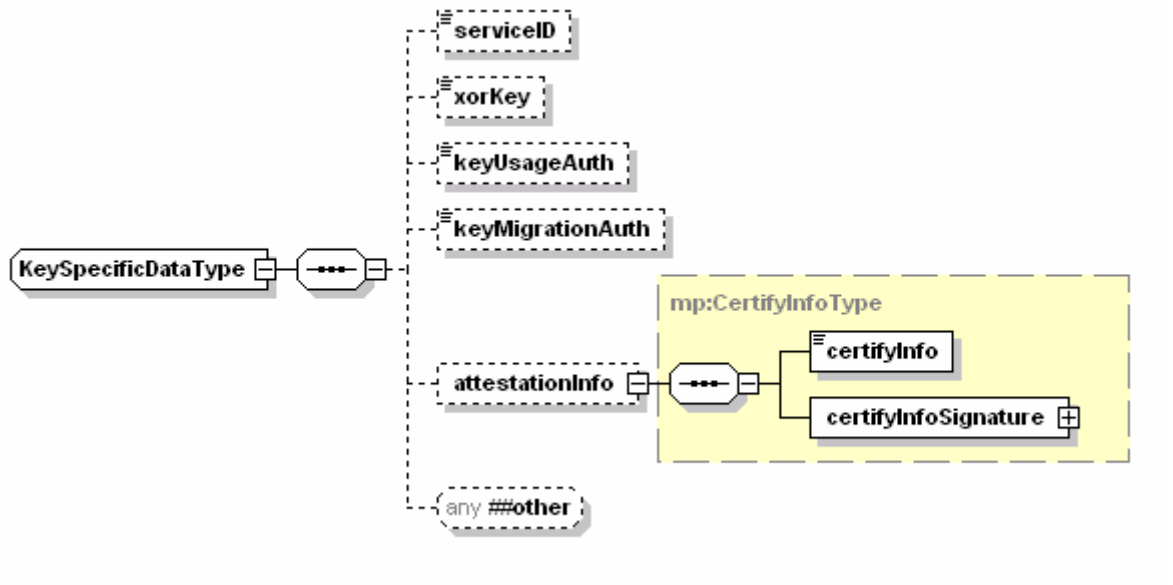


Figure 3 – KeySpecificDataType Schema

In the case the migrating key is a MK, an AIK may not be used to certify the MK as the TPM disallows such use of an AIK. A NMK or other MK may however be used to certify the MK. The value of such certification is questionable however, since the NMK or certifying MK may be used to sign any arbitrary data and thus provides little proof that the migrating key is actually a TPM key. This approach may still be valuable in cases when a secure operating system is available on the platform.

The KeySpecificDataType provides for the following elements:

Element	Type	Required	Reference
serviceID	String	N	Identifier of service which is used to create KeySpecificData. Useful for identifying the required service needed to reconstruct the relevant KeySpecific data on a key download operation.
xorKey	Base64binary	N	A TPM generated random value which is used to provide confidentiality for the migration blob (tssBlob) from the Migration Authority.
keyUsageAuth	Base64binary	N	{Plaintext} form of UsageAuth as defined in TPM_STORE_ ASYMKEY or TPM_MIGRATE_ASYMKEY
keyMigrationAuth	Base64binary	N	{Plaintext} form of migrationAuth as defined in TPM_STORE_ASYMKEY
attestationInfo	CertifyInfo	N	For a CMK, this proves that the key is TPM hardware resident, and the TPM_CERTIFY_INFO2 structure should be passed along with the AIK Credential. For an MK, the value of the certifyInfo structure is questionable, but if provided, should pass a TPM_CERTIFY_INFO structure along with the AIK Credential.
Other	Base64binary	N	Vendor specific data.

Table 8 - KeySpecificDataType

TPM_CERTIFY_INFO and TPM_CERTIFY_INFO2 structures are defined in the TPM v1.1b and v1.2 specifications.

4.2.1.7 encApplicationSpecificDataType & encApplicationSpecificDataKey

The encApplicationSpecificDataType is the xml-enc encrypted ApplicationSpecificDataType. The encApplicationSpecificDataKey is the xml-enc encrypted symmetric key which is used to protect the ApplicationSpecificData.

Element	Type	Required	Reference
encApplicaitonSpecificDataType	xenc:EncryptedDataType	N	Xml-enc encrypted ApplicationSpecificData.
encApplicationSpecificDataKey	xenc:EncryptedKeyType	N	Xml-enc encrypted symmetric key which is used to protect ApplicationSpecificData.

Table 9

4.2.1.8 ApplicationSpecificDataType

Certificates and any relevant CSP container data is optionally available in the ApplicationSpecificData element. The ApplicationSpecificDataType has the following schema:

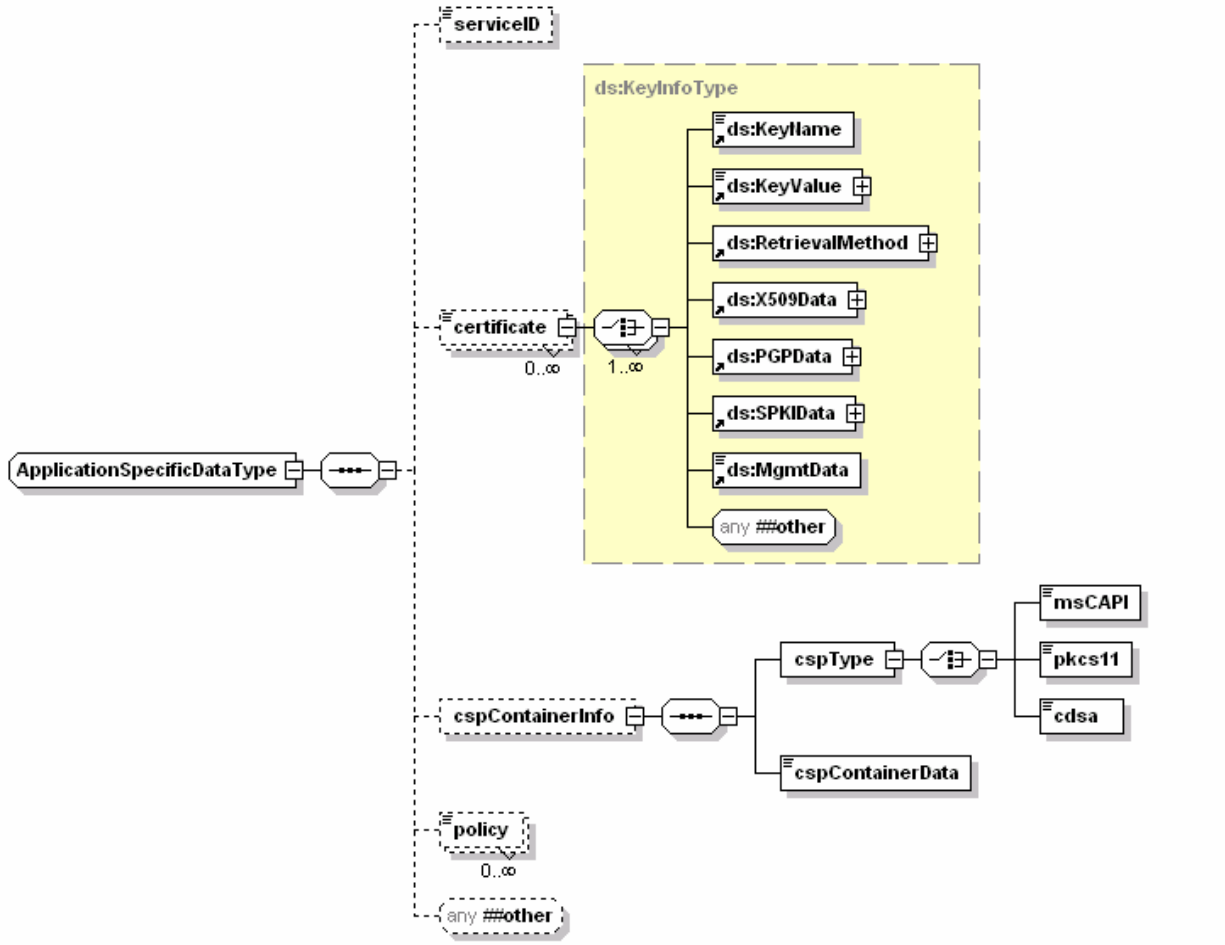


Figure 4 – ApplicationSpecificDataType Schema

The ApplicationSpecificDataType provides for the following elements:

Element	Type	Required	Reference
serviceID	String	N	Identifier of service which is used to create ApplicationSpecificData. Useful for identifying the required service needed to reconstruct the relevant Application Specific Data on a key download operation.
Certificate	ds:KeyInfo	N	Structure to support storing of certificate which corresponds to private key being migrated
cspContainerInfo	ComplexType	N	CSP (or equivalent) information which allows a process or service to import a TPM key reference into the provider.
cspType	String	Y(d)	Within cspContainerInfo - Text label which indicates the type of csp that the key was registered with and where associated migration information came from.
cspContainerData	Base64binary	Y(d)	Within cspContainerInfo - Encoded data which is relevant to the reconstruction of the key data in the identified CSP.
policy	Base64binary	N	Policies assigned to use of key or certificate
other	Base64binary	N	Vendor specific data.

Table 10 - ApplicationSpecificDataType

5 LMS Functions

All of Section 5 is INFORMATIVE only.

The interface from local applications to the LMS is intended to be vendor specific and does not affect interoperability of migration services. The LMS functions described in this section are provided as an example of the types of functions that could be provided to expose a complete migration service.

5.1 LMS_RegisterMPDataEncryptionKey

Start of informative comment:

LMS_RegisterMPDataEncryptionKey registers a public key within the LMS to allow the TPM user to specify which encryption key LMS will use in order to provide for the asymmetric encryption of the symmetric key which protects the KeySpecificData and ApplicationSpecificData elements of a MP.

Encryption of the KeySpecificData and ApplicationSpecificData is optional. If confidentiality is selected, the user may choose to encrypt with a public key known to the Migration Authority to ensure that the MP may be recovered by the MA, or the user may choose to encrypt with a public key whose private portion is known only to the user to ensure that only the user may recover the SpecificData.

Conceptually, the symmetric key may also be registered twice, once with an MA pub key and a second time with a user pub key in order that the MA or the user may independently recover the keys.

End of informative comment.

5.2 LMS_UnregisterMPDataEncryptionKey

Start of informative comment:

Removes the Migration Package data encryption key from LMS.

End of informative comment.

5.3 LMS_QueryMPDataEncryptionKey

Start of informative comment:

LMS_QueryMPDataEncryptionKey enables the user to determine which MP Data Encryption Keys are currently registered for the platform.

End of informative comment.

5.4 LMS_RegisterMigrationAuthority

Start of informative comment:

LMS_RegisterMigrationAuthority enables the user to register a Migration Authority as a valid Migration Authority for the platform. The process of registering a migration authority will create the necessary migration ticket.

Inputs to LMS_RegisterMigrationAuthority include the MA public key (for creation of a migration ticket) and the MA URI (to locate the web service / WSDL). The desired Migration Authority for a migration operation must be chosen from the LMS database of registered Migration Authority.

End of informative comment.

5.5 LMS_UnregisterMigrationAuthority

Start of informative comment:

Removes the Migration Authority and corresponding migration ticket for a Migration Authority from LMS.

End of informative comment.

5.6 LMS_QueryMigrationAuthority

Start of informative comment:

LMS_QueryPeerToPeerMigrationTarget enables the user to determine which Peer-To-Peer Migration Target keys are currently registered for the platform.

End of informative comment.

5.7 LMS_RegisterPeerToPeerMigrationTarget

Start of informative comment:

LMS_RegisterPeerToPeerMigrationTarget enables the user to register the key of a target TPM platform as a valid migration destination for the platform. The process of registering the target TPM key will create the necessary migration ticket.

Peer-To-Peer migration uses the TPM's Rewrap migration scheme and provides for direct migration to a target without an intermediary..

End of informative comment.

5.8 LMS_UnregisterPeerToPeerMigrationTarget

Start of informative comment:

Removes the key and corresponding migration ticket for a target TPM key from LMS.

End of informative comment.

5.9 LMS_QueryPeerToPeerMigrationTarget

Start of informative comment:

LMS_QueryMigrationAuthority enables the user to determine which Migration Authorities are currently registered for the platform.

End of informative comment.

5.10 LMS_RegisterMigrationSelectionAuthority

Start of informative comment:

LMS_RegisterMigrationSelectionAuthority enables the user to register a Migration Selection Authority as a valid Migration Selection Authority for the platform. The process of registering a migration authority will create the necessary migration ticket.

The desired Migration Selection Authority for a migration operation must be chosen from the LMS database of registered Migration Selection Authorities.

End of informative comment.

5.11 LMS_UnregisterMigrationSelectionAuthority

Start of informative comment:

Removes the key and corresponding structures for a Migration Selection Authority from LMS.

End of informative comment.

5.12 LMS_QueryMigrationSelectionAuthority

Start of informative comment:

LMS_QueryMigrationSelectionAuthority enables the user to determine which Migration Selection Authorities are currently registered for the platform.

End of informative comment.

5.13 LMS_ComposeMigrationPackage

Start of informative comment:

LMS_ComposeMigrationPackage creates a valid migration package and prepares the MP for the UPLOAD operation.

End of informative comment.

5.14 LMS_DecomposeMigrationPackage

Start of informative comment:

LMS_DecomposeMigrationPackage parses an MP which has been retrieved from a DOWNLOAD operation. The MP is verified by the client and any encrypted elements are decrypted for processing into the constituent components.

End of informative comment.

5.15 LMS_MAUpload

Start of informative comment:

LMS_MAUpload uploads a MP to the MA or to a Peer-To-Peer platform.

End of informative comment.

5.16 LMS_MADownload

Start of informative comment:

LMS_MADownload retrieves a MP from an MA or a Peer-To-Peer TPM platform.

End of informative comment.

5.17 LMS_MAQuery

Start of informative comment:

LMS_MAQuery retrieves a list of all MPs available for download from a MP or Peer-To-Peer platform.

End of informative comment.

5.18 LMS_MAConditionalQuery

Start of informative comment:

LMS_MAConditionalQuery retrieves a list of all MPs available for download from a MP or a Peer-To-Peer platform that for which one or more conditions are valid.

End of informative comment.

5.19 LMS_MADelete

Start of informative comment:

LMS_MADelete requests a MA to delete a MP from its persistent storage. Execution of this command by the MA is dependent upon the policy implementation.

End of informative comment.

5.20 LMS_Export

Start of informative comment:

LMS_Export returns a MP to the calling application. Responsibility to store or transport the MP to another entity resides with the calling application. This call is intended to support out-of-band transfer of MPs (e.g. email of a MP to another entity).

End of informative comment.

5.21 LMS_Import

Start of informative comment:

LMS_Import accepts a MP from the calling application.

End of informative comment.

5.22 LMS_GetMAServiceInfo

Start of informative comment:

LMS_GetMAServiceInfo returns an XML string to the calling application. This string provides any information the MA chooses to describe its service, including URI, policies and migration practice statement. This call is intended to provide informative human readable data on the services provided by an MA.

End of informative comment.

5.23 LMS_GetMAKey

Start of informative comment:

LMS_GetMAKey returns the MA certificate containing the MA Public Key to the calling application. The MA Public Key is used as an input to LMS_RegisterMigrationAuthority.

End of informative comment.

5.24 LMS_RegisterKeyData

Start of informative comment:

LMS_RegisterKeyData provides a mechanism for local applications, CSPs, and TCG-related services to register a key or key-related data with the LMS for future migration.

End of informative comment.

5.25 LMS_GetKeyData

Start of informative comment:

LMS_GetKeyData provides a mechanism for local applications, CSPs, and TCG-related services to retrieve a key or key-related data from the LMS after completion of a key migration.

End of informative comment.

5.26 LMS_RetrieveData

Start of informative comment:

LMS_RetrieveData prompts the LMS that either a new key was created or key information was altered, and that the LMS should attempt to retrieve updated key data in preparation of a future archive. This only applies to Active LMS models.

End of informative comment.

5.27 Sequence Diagrams

Figure 5 demonstrates the potential interactions between the LMS and other local components whereby the LMS actively retrieves data required to create a MP. The MP is then ready for transfer to a MA utilizing the LMS_MAUpload call. LMS_DecomposeMigrationPackage is the reverse.

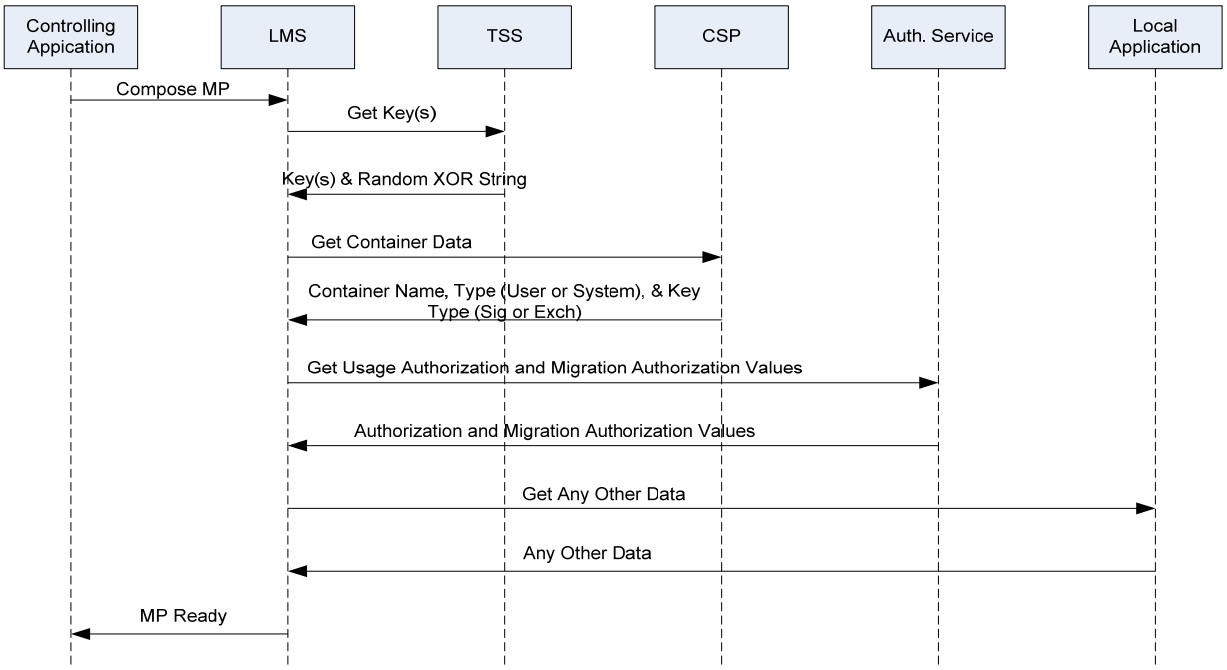


Figure 5: Migration Package Construction - LMS Actitve

Figure 6 demonstrates an alternate (yet equally valid) set of potential interactions between the LMS and other local components whereby the data required to create a MP is registered with the LMS by local applications and services and LMS does not need to actively collect the data. The MP is then ready for transfer to a MA utilizing the LMS_MAUpload call. LMS_DecomposeMigrationPackage is the reverse.

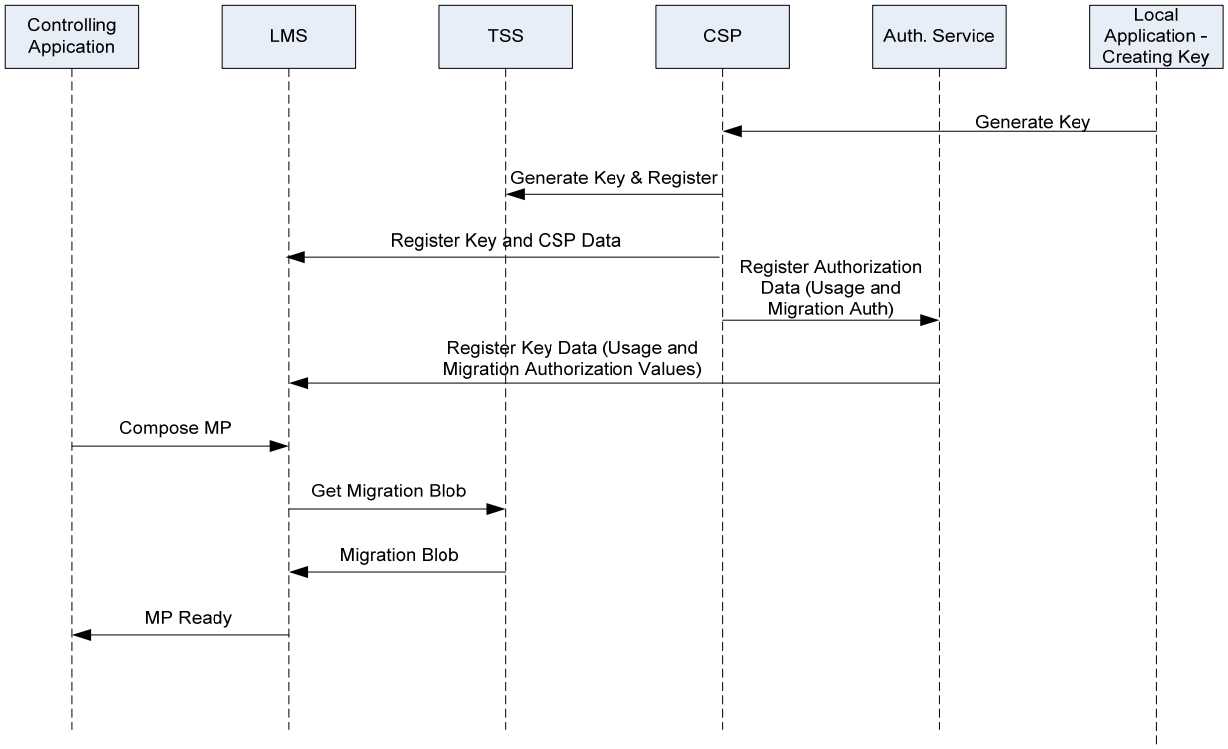


Figure 6: Migration Package Construction - LMS Passive

Figure 7 demonstrates the potential interactions between the LMS and other local components whereby the LMS requests the TSS to convert the migration blob and actively writes associated to the components. The key is then ready for use on the platform.

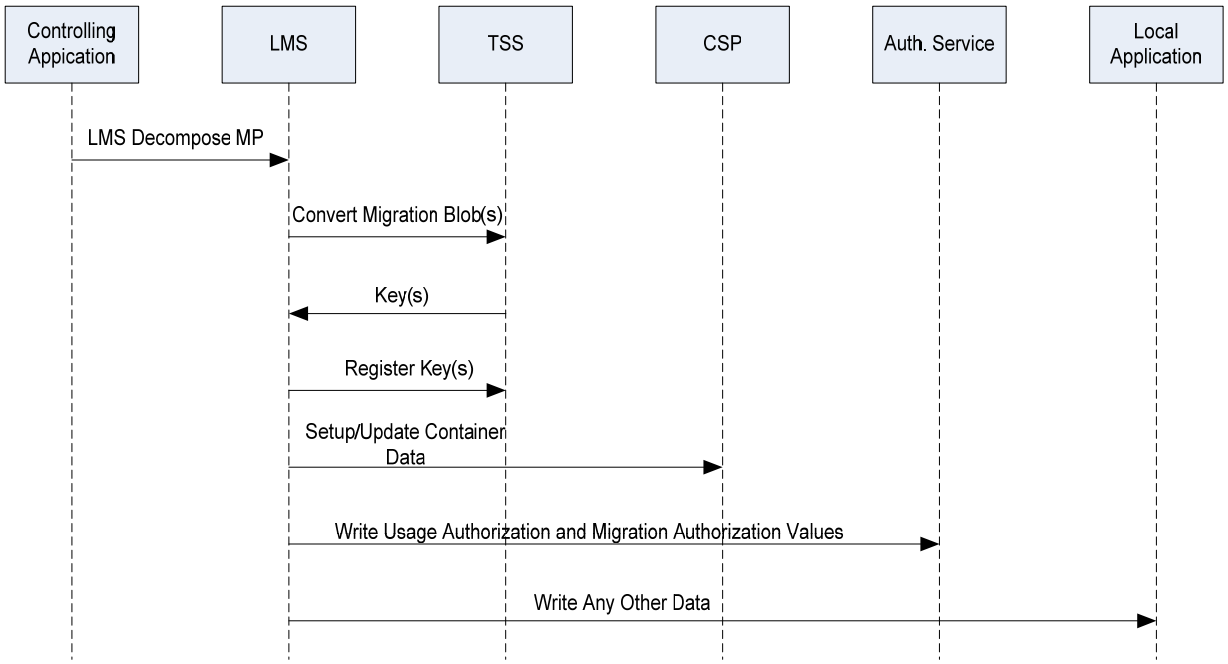


Figure 7: Migration Package Consumption – LMS Active

Figure 8 demonstrates the potential interactions between the LMS and other local components whereby the LMS requests the TSS to convert the migration blob, sends notice to the components of availability of new key data, and the components retrieve the data whenever they are ready. The key is then ready for use on the platform.

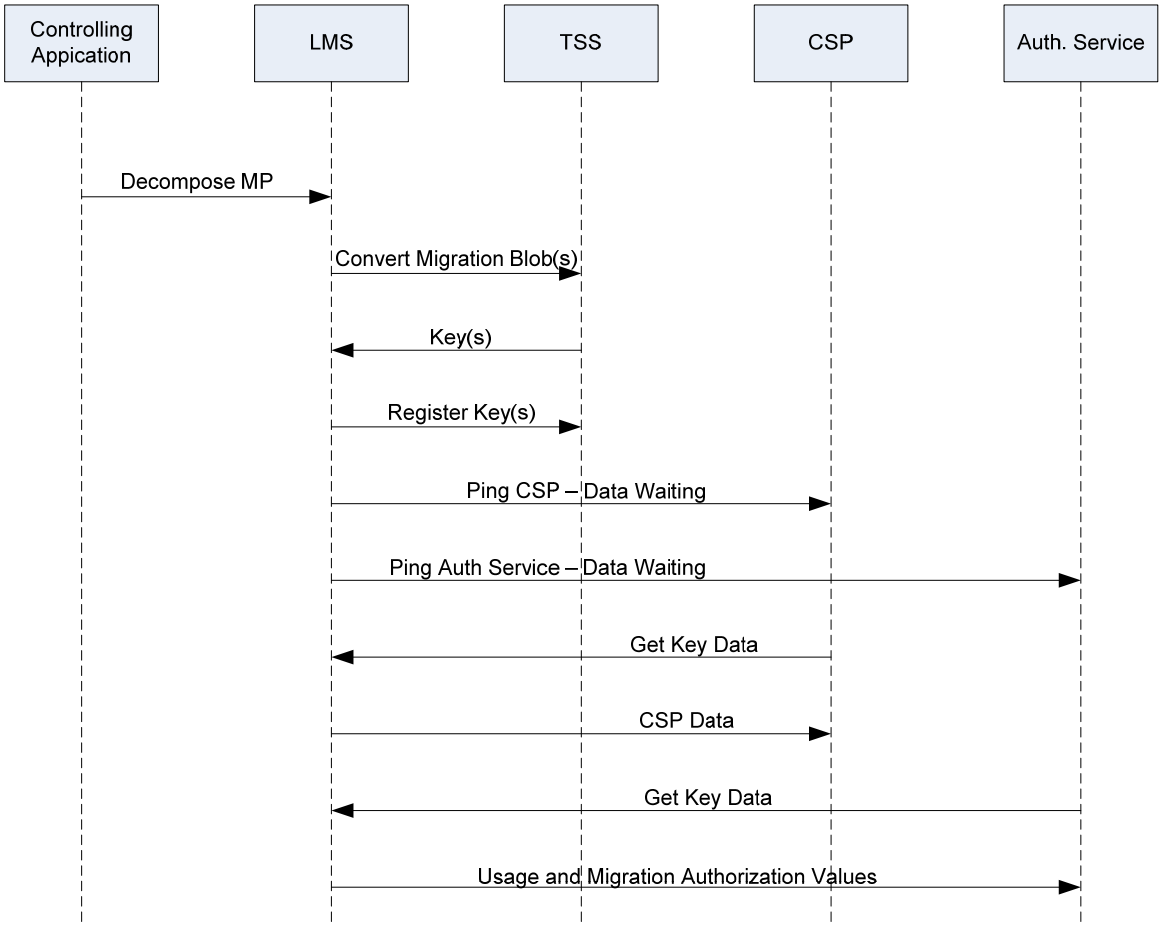


Figure 8: Migration Package Consumption - LMS Passive

6 Service Models

6.1 Vaulted Migration Service

Migration services are seen as a client initiated interaction with a service provider (SP). Nominally the SP is a Migration Authority where the keys are vaulted, or escrowed, in a secure location for use by a valid authenticated client. Once authenticated, the client may upload, download, or query the Migration Authority about Migration Packages (MPs). Additionally, it may be possible for the client to request deletion of a MP from a vaulted MA environment.

The MP should contain any associated information which the operating environment determines should be archived with the key.

1. Random XOR String – If an enterprise expects to be able to recover this key in the event the user ceases to be available / exist, then it may require storage of the Random XOR string with recovery under suitable policy restrictions. For privacy and legal purposes, this option may not be available. If not passed, then for migrate mode it must be passed out of band to the eventual target TPM (ok for restoration of key for original user on another platform).
2. Usage Auth Value – If an enterprise expects to be able to use a recovered key in the even the holder of the key usage authorization value ceases to be available / exist, then it may require storage of the usage auth value with recovery under suitable policy restrictions. Similarly, if there is an expectation that the future target TPM user be able to use the key, then the usage auth value needs to be transferred in some mechanism; out-of-band is also an option.
3. Migration Auth Value – If the user wishes to allow the future target TPM to migrate the key in the future (especially in case where target parent key is non-migratory), then the migration auth value needs to be transferred via some mechanism; out-of-band is also an option.
4. Attestation Information – If the user wishes to prove to either the Migration Authority or to the eventual recipient that this key was generated on a genuine TPM, then the user should include attestation information for that key.
5. Certificates – If the user intends to be the recipient of a restoration of the keys, the user will want to ensure any associated certificates also get backed up. In the case when a key may be intended for another party, then the user may choose to withhold the certificate and force the recipient to request their own certificate for use of that key.
6. CSP Container Information – In most current implementations, applications do not use TPM keys by interfacing with the TSS directly, but rather interface with well known cryptographic interfaces such as MS CAPI or PKCS #11. If the user and application expect to use the key in the same way on the target platform, they may need some data regarding the CSP container in order to adequately recreate the environment on the target platform.
7. Policies – If new policies are not going to be created for a key on the target platform, then currently existing policies should travel with the key.
8. Children Keys – If the migrating key is the parent of other keys (i.e. it is a storage key), then the user is trying to backup the children of that key and the children of interest should be included in the Migration Package.

It is the responsibility of the client to be able to properly format any optional MP data on an upload, download, or query operation.

6.1.1 Provider Entity (PE)

The Provider Entity is always the Migration Authority.

6.1.2 Requestor Entity (RE)

The Requestor entity is the TCG compliant client. The RE may be requesting to upload a key, download a key, or query the PE about which keys it has.

6.1.3 Entity Relationship

The RE will ALWAYS initiate vaulted migration operations.

6.1.4 Service Tasks

(Note: A service task is an action or combination of actions that is associated with a particular goal state. Performing the task involves executing the actions, and is intended to achieve a particular goal state.)

- Authenticate – prove valid possession of enabled and activated TPM. For a CMK, an AIK Credential and CertifyInfo provide proof. For a MK, an AIK-certified NMK can be used in an SSL session to provide proof. For a CMK upload, prove key exists within a TPM. For a download, prove that target parent storage key is from valid TPM and for CMKs that parent key is non-migratable. Determine existence of valid RE-owned vault at Migration Authority. Receive token valid for 1 or more MA operations.
- Upload – RE sends a MP to PE
- Download – retrieve a MP from PE for RE
- Query – interrogate the PE as to what valid MPs exist for this RE.
- Delete – RE requests deletion of MP by PE
- Get MA Service Information – RE requests service information for this PE
- Get MA Key – RE requests migration key for this PE

6.1.5 Policies

The RE can query the PE to determine if the necessary MP options are available, required, or not supported.

6.2 Transitory Migration Service Description

A Transitory Migration Service is similar to a Vaulted Migration Service, but fundamentally differs in that the MP is not intended to reside at the service provider, but rather is intended to be forwarded to another recipient. All of the same security and authentication requirements apply as for the Vaulted Migration Service. However, the client has no ability to delete MPs at the service provider since it does not permanently store them.

6.2.1 Provider Entity (PE)

The Provider Entity is always the Migration Authority.

6.2.2 Requestor Entity (RE)

The Requestor entity is the TCG compliant client. The RE may be requesting to upload a key, download a key, or query the PE to see if it has any keys are available for download.

6.2.3 Entity Relationship

The RE will ALWAYS initiate transitory migration operations.

6.2.4 Service Tasks

- Authenticate – prove valid possession of enabled and activated TPM. For a CMK, an AIK Credential and CertifyInfo provide proof. For a MK, an AIK-certified NMK can be used in an SSL session to provide proof. For a CMK upload, prove key exists within a TPM. For a download, prove that target parent storage key is from valid TPM and for CMKs that parent key is non-migratable. Receive token valid for 1 or more MA operations.
- Upload – RE sends a MP to PE

- Download – retrieve a MP from PE for RE
- Query – interrogate the PE as to what valid MPs exist for this RE.
- Get MA Service Information – RE requests service information for this PE
- Get MA Key – RE requests migration key for this PE

6.3 Direct Migration Service Description

A Direct Migration provides the Peer-to-Peer key migration capability. It is suitable for making a copy of a client's keys on another platform, but generally is intended to enable transfer of a single key from one client to another. As such, communication is from the LMS on one client to the LMS on a second client and the PE and RE roles are reversible.

6.3.1 Provider Entity (PE)

The Provider Entity is a TCG compliant client whose local environment may be set up to provide client or server migration services.

6.3.2 Requestor Entity (RE)

The Requestor entity is the TCG compliant client. The RE may be requesting to upload a key, download a key, or query the PE to see if it has any keys are available for download.

6.3.3 Entity Relationship

The RE will ALWAYS initiate transitory migration operations.

6.3.4 Service Tasks

- Authenticate – prove valid possession of enabled and activated TPM. For a CMK, an AIK Credential and CertifyInfo provide proof. For a MK, an AIK-certified NMK can be used in an SSL session to provide proof. For a CMK upload, prove key exists within a TPM. For a download, prove that target parent storage key is from a valid TPM and for CMKs that the parent key is non-migratable.
- Upload – RE sends a MP to PE
- Download – retrieve a MP from PE for RE
- Query – interrogate the PE as to what valid MPs exist for this RE.
- Get MA Service Information – RE requests service information for this PE
- Get MA Key – RE requests migration key for this PE

7 Migration WSDL

The following WSDL provides a reference template for a Migration Authority service. The primary functions provided are: Uploading a MP, Querying the MA for available MPs, Downloading a MP, Deleting a MP from the MP's available list, requesting the MA Public Key and requesting human readable information describing the MA's service. Of course, a MA may choose to implement a subset of these functions or more functions; however it is expected that services supporting these functions are exposed in this manner in order to allow for LMS interoperability. Implementers of a Migration Authority service will need to customize their WSDL file to refer to the URL of their service by replacing the Migration Service URL in the supplied WSDL file.

Efforts have been made to ensure the WSDL is WS-I compliant. The tools that generated the WSDL does not guarantee WS-I compliance and the WSDL defined in this document is not guaranteed to be WS-I 1.1 compliant. However, the WSDL has been checked by several tools for WS-I compliance and has passed successfully. In addition, the implementation follows design guidelines published by Microsoft for building interoperable web services: WS-I Basic Profile 1.0.

Note: The WSDL file MigrationService_1-00_1-00.wsdl may be located at:

http://www.trustedcomputinggroup.org/XML/WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
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://www.trustedcomputinggroup.org/XML/WSDL/MigrationService_1-00_1-00"
xmlns:tm=http://microsoft.com/wsdl/mime/textMatching/
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://www.trustedcomputinggroup.org/XML/WSDL/MigrationService_1-00_1-00"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    <wsdl:types>
        <s:schema elementFormDefault="qualified"
          targetNamespace="http://www.trustedcomputinggroup.org/XML/WSDL/MigrationService_1-00_1-00">
            <s:element name="MigrationServiceInformation">
                <s:complexType/>
            </s:element>
            <s:element name="MigrationServiceInformationResponse">
                <s:complexType>
                    <s:sequence>
                        <s:element minOccurs="1" maxOccurs="1" name="MigrationServiceInformationResult"
                          type="tns:ReturnCode"/>
                        <s:element minOccurs="0" maxOccurs="1" name="serviceInformation">
                            <s:complexType mixed="true">
                                <s:sequence>
                                    <s:any/>
                                </s:sequence>
                            </s:complexType>
                        </s:element>
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:simpleType name="ReturnCode">
                <s:list>
                    <s:simpleType>
                        <s:restriction base="s:string">
                            <s:enumeration value="Succeed"/>
                            <s:enumeration value="Failed"/>
                            <s:enumeration value="NotImplemented"/>
                            <s:enumeration value="ServerNotAvailable"/>
                        </s:restriction>
```

```
                </s:simpleType>
            </s:list>
        </s:simpleType>
        <s:element name="Upload">
            <s:complexType>
                <s:sequence>
                    <s:element minOccurs="0" maxOccurs="1" name="migrationPackage">
                        <s:complexType mixed="true">
                            <s:sequence>
                                <s:any/>
                            </s:sequence>
                        </s:complexType>
                    </s:element>
                    <s:element minOccurs="1" maxOccurs="1" name="transitoryFlag" type="s:boolean"/>
                    <s:element minOccurs="1" maxOccurs="1" name="targetIdentificationTokenType"
                      type="tns:TargetIdentificationTokenType"/>
                    <s:element minOccurs="0" maxOccurs="1" name="targetIdentificationToken"
                      type="s:base64Binary"/>
                </s:sequence>
            </s:complexType>
        </s:element>
        <s:simpleType name="TargetIdentificationTokenType">
            <s:list>
                <s:simpleType>
                    <s:restriction base="s:string">
                        <s:enumeration value="PlubicKey"/>
                        <s:enumeration value="Certificate"/>
                        <s:enumeration value="URI"/>
                        <s:enumeration value="Other"/>
                    </s:restriction>
                </s:simpleType>
            </s:list>
        </s:simpleType>
        <s:element name="UploadResponse">
            <s:complexType>
                <s:sequence>
                    <s:element minOccurs="1" maxOccurs="1" name="UploadResult" type="tns:ReturnCode"/>
                    <s:element minOccurs="0" maxOccurs="1" name="migrationPackageId" type="s:string"/>
                </s:sequence>
            </s:complexType>
        </s:element>
        <s:element name="Download">
            <s:complexType>
                <s:sequence>
                    <s:element minOccurs="0" maxOccurs="1" name="migrationPackageId" type="s:string"/>
                    <s:element minOccurs="0" maxOccurs="1" name="parentPubKey" type="s:base64Binary"/>
                    <s:element minOccurs="0" maxOccurs="1" name="attestationInfo" type="tns:AttestationInfo"/>
                    <s:element minOccurs="0" maxOccurs="1" name="encryptionKey" type="s:base64Binary"/>
                </s:sequence>
            </s:complexType>
        </s:element>
        <s:complexType name="AttestationInfo">
            <s:sequence>
                <s:element minOccurs="0" maxOccurs="1" name="certifyInfo" type="s:base64Binary"/>
                <s:element minOccurs="0" maxOccurs="1" name="signatureValue" type="s:base64Binary"/>
                <s:element minOccurs="0" maxOccurs="1" name="aikCredential" type="s:base64Binary"/>
            </s:sequence>
        </s:complexType>
        <s:element name="DownloadResponse">
            <s:complexType>
                <s:sequence>
                    <s:element minOccurs="1" maxOccurs="1" name="DownloadResult" type="tns:ReturnCode"/>
                    <s:element minOccurs="0" maxOccurs="1" name="migrationPackage">
                        <s:complexType mixed="true">
                            <s:sequence>
                                <s:any/>
```

```
                        </s:sequence>
                    </s:complexType>
                </s:element>
                <s:element minOccurs="0" maxOccurs="1" name="tssKeyBlob" type="s:base64Binary"/>
            </s:sequence>
        </s:complexType>
    </s:element>
    <s:element name="Query">
        <s:complexType>
            <s:sequence>
                <s:element minOccurs="1" maxOccurs="1" name="recCount" type="s:long"/>
            </s:sequence>
        </s:complexType>
    </s:element>
    <s:element name="QueryResponse">
        <s:complexType>
            <s:sequence>
                <s:element minOccurs="1" maxOccurs="1" name="QueryResult" type="tns:ReturnCode"/>
                <s:element minOccurs="0" maxOccurs="1" name="migrationPakcageItemList"
                  type="tns:ArrayOfMigrationPackageListItem"/>
            </s:sequence>
        </s:complexType>
    </s:element>
    <s:complexType name="ArrayOfMigrationPackageListItem">
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="unbounded" name="MigrationPackageListItem"
              nillable="true" type="tns:MigrationPackageListItem"/>
        </s:sequence>
    </s:complexType>
    <s:complexType name="MigrationPackageListItem">
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="packageID" type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="description" type="s:string"/>
            <s:element minOccurs="1" maxOccurs="1" name="uploadTime" type="s:dateTime"/>
            <s:element minOccurs="1" maxOccurs="1" name="dataSize" type="s:long"/>
            <s:element minOccurs="0" maxOccurs="1" name="keyUUID" type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="ownerName" type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="parentUUID" type="s:string"/>
            <s:element minOccurs="0" maxOccurs="1" name="certHashList" type="tns:ArrayOfCertificateHash"/>
        </s:sequence>
    </s:complexType>
    <s:complexType name="ArrayOfCertificateHash">
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="unbounded" name="CertificateHash" nillable="true"
              type="tns:CertificateHash"/>
        </s:sequence>
    </s:complexType>
    <s:complexType name="CertificateHash">
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="hashValue" type="s:base64Binary"/>
        </s:sequence>
    </s:complexType>
    <s:element name="QueryByXPath">
        <s:complexType>
            <s:sequence>
                <s:element minOccurs="0" maxOccurs="1" name="migrationPackageXPath" type="s:string"/>
            </s:sequence>
        </s:complexType>
    </s:element>
    <s:element name="QueryByXPathResponse">
        <s:complexType>
            <s:sequence>
                <s:element minOccurs="1" maxOccurs="1" name="QueryByXPathResult"
                  type="tns:ReturnCode"/>
                <s:element minOccurs="0" maxOccurs="1" name="queryResult">
                    <s:complexType mixed="true">
```

```
                                <s:sequence>
                                    <s:any/>
                                </s:sequence>
                            </s:complexType>
                        </s:element>
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:element name="Delete">
                <s:complexType>
                    <s:sequence>
                        <s:element minOccurs="0" maxOccurs="1" name="mpUuidList" type="tns:ArrayOfString"/>
                    </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="DeleteResponse">
                <s:complexType>
                    <s:sequence>
                        <s:element minOccurs="1" maxOccurs="1" name="DeleteResult" type="tns:ReturnCode"/>
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:element name="GetMAKey">
                <s:complexType/>
            </s:element>
            <s:element name="GetMAKeyResponse">
                <s:complexType>
                    <s:sequence>
                        <s:element minOccurs="1" maxOccurs="1" name="GetMAKeyResult" type="tns:ReturnCode"/>
                        <s:element minOccurs="0" maxOccurs="1" name="maKey"
                          type="tns:ArrayOfMigrationAuthorityKey"/>
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:complexType name="ArrayOfMigrationAuthorityKey">
                <s:sequence>
                    <s:element minOccurs="0" maxOccurs="unbounded" name="MigrationAuthorityKey" nillable="true"
                      type="tns:MigrationAuthorityKey"/>
                </s:sequence>
            </s:complexType>
            <s:complexType name="MigrationAuthorityKey">
                <s:sequence>
                    <s:element minOccurs="0" maxOccurs="1" name="maCertificate" type="s:base64Binary"/>
                    <s:element minOccurs="0" maxOccurs="1" name="maAttestationInfo" type="s:base64Binary"/>
                </s:sequence>
            </s:complexType>
        </s:schema>
    </wsdl:types>
    <wsdl:message name="MigrationServiceInformationSoapIn">
        <wsdl:part name="parameters" element="tns:MigrationServiceInformation"/>
    </wsdl:message>
    <wsdl:message name="MigrationServiceInformationSoapOut">
        <wsdl:part name="parameters" element="tns:MigrationServiceInformationResponse"/>
    </wsdl:message>
    <wsdl:message name="UploadSoapIn">
        <wsdl:part name="parameters" element="tns:Upload"/>
    </wsdl:message>
    <wsdl:message name="UploadSoapOut">
        <wsdl:part name="parameters" element="tns:UploadResponse"/>
    </wsdl:message>
    <wsdl:message name="DownloadSoapIn">
```

```
        <wsdl:part name="parameters" element="tns:Download"/>
    </wsdl:message>
    <wsdl:message name="DownloadSoapOut">
        <wsdl:part name="parameters" element="tns:DownloadResponse"/>
    </wsdl:message>
    <wsdl:message name="QuerySoapIn">
        <wsdl:part name="parameters" element="tns:Query"/>
    </wsdl:message>
    <wsdl:message name="QuerySoapOut">
        <wsdl:part name="parameters" element="tns:QueryResponse"/>
    </wsdl:message>
    <wsdl:message name="QueryByXPathSoapIn">
        <wsdl:part name="parameters" element="tns:QueryByXPath"/>
    </wsdl:message>
    <wsdl:message name="QueryByXPathSoapOut">
        <wsdl:part name="parameters" element="tns:QueryByXPathResponse"/>
    </wsdl:message>
    <wsdl:message name="DeleteSoapIn">
        <wsdl:part name="parameters" element="tns:Delete"/>
    </wsdl:message>
    <wsdl:message name="DeleteSoapOut">
        <wsdl:part name="parameters" element="tns:DeleteResponse"/>
    </wsdl:message>
    <wsdl:message name="GetMAKeySoapIn">
        <wsdl:part name="parameters" element="tns:GetMAKey"/>
    </wsdl:message>
    <wsdl:message name="GetMAKeySoapOut">
        <wsdl:part name="parameters" element="tns:GetMAKeyResponse"/>
    </wsdl:message>
    <wsdl:portType name="MigrationServiceSoap">
        <wsdl:operation name="MigrationServiceInformation">
            <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Return Migration Authority information XML
              string. Migration Server can selectively implement some of the methods defined in this interface. The XML
              string provides the MA URL, MA Practice Statement and any other human readable information of the MA's
              choosing to describe its service. It will return ReturnCode.NotImplemented if the method is not
              implemented.</documentation>
            <wsdl:input message="tns:MigrationServiceInformationSoapIn"/>
            <wsdl:output message="tns:MigrationServiceInformationSoapOut"/>
        </wsdl:operation>
        <wsdl:operation name="Upload">
            <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Upload the migration package XML to the
              Migration Authority. migrationPackageId will contain the package ID if the operation is succeed. If the
              TransitoryFlag is True, then the Migration Authority will delete the MP after downloading to the target
              platform.</documentation>
            <wsdl:input message="tns:UploadSoapIn"/>
            <wsdl:output message="tns:UploadSoapOut"/>
        </wsdl:operation>
        <wsdl:operation name="Download">
            <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Download the migration pacakge by package ID.
              The server will verify if the requester has download privilege for the specified package. tssKeyBlob is for the
              case when the MP was signed by a local key. Then the entire old MP must be returned to the target so that
              the integrity can be verified, and the MA must additionally pass down the newly encrypted
              key.</documentation>
            <wsdl:input message="tns:DownloadSoapIn"/>
            <wsdl:output message="tns:DownloadSoapOut"/>
        </wsdl:operation>
        <wsdl:operation name="Query">
            <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">The function returns the downloadable migration
              package list for the requesting user. The list is in the decending ordered of uploadTime.</documentation>
            <wsdl:input message="tns:QuerySoapIn"/>
            <wsdl:output message="tns:QuerySoapOut"/>
        </wsdl:operation>
        <wsdl:operation name="QueryByXPath">
            <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Query the downloadable migration packages by
              specifing  the migration packages XPath. The returning XML root node name is
              migrationPackageList.</documentation>
```

```
            <wsdl:input message="tns:QueryByXPathSoapIn"/>
            <wsdl:output message="tns:QueryByXPathSoapOut"/>
        </wsdl:operation>
        <wsdl:operation name="Delete">
            <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Delete migration packages by IDs. The server
              either deletes all the packages successfully or does nothing. The server will verify if the requester has
              deletion privilege for all the specified packages.</documentation>
            <wsdl:input message="tns:DeleteSoapIn"/>
            <wsdl:output message="tns:DeleteSoapOut"/>
        </wsdl:operation>
        <wsdl:operation name="GetMAKey">
            <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">One or more MA certificates (DSIG) containing
              the MA Public key. Also optional for each certificate is maAttestationInfo (i.e. if the MA key is a TPM
              key).</documentation>
            <wsdl:input message="tns:GetMAKeySoapIn"/>
            <wsdl:output message="tns:GetMAKeySoapOut"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="MigrationServiceSoap" type="tns:MigrationServiceSoap">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
        <wsdl:operation name="MigrationServiceInformation">
            <soap:operation soapAction="http://www.trustedcomputinggroup.org/XML/WSDL/MigrationService_1-00_1-
00/MigrationServiceInformation" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="Upload">
            <soap:operation soapAction="http://www.trustedcomputinggroup.org/XML/WSDL/MigrationService_1-00_1-
00/Upload" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="Download">
            <soap:operation soapAction="http://www.trustedcomputinggroup.org/XML/WSDL/MigrationService_1-00_1-
00/Download" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="Query">
            <soap:operation soapAction="http://www.trustedcomputinggroup.org/XML/WSDL/MigrationService_1-00_1-
00/Query" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="QueryByXPath">
            <soap:operation soapAction="http://www.trustedcomputinggroup.org/XML/WSDL/MigrationService_1-00_1-
00/QueryByXPath" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
```

```
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="Delete">
            <soap:operation soapAction="http://www.trustedcomputinggroup.org/XML/WSDL/MigrationService_1-00_1-
00/Delete" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="GetMAKey">
            <soap:operation soapAction="http://www.trustedcomputinggroup.org/XML/WSDL/MigrationService_1-00_1-
00/GetMAKey" style="document"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="MigrationService">
        <documentation xmlns="http://schemas.xmlsoap.org/wsdl/"/>
        <wsdl:port name="MigrationServiceSoap" binding="tns:MigrationServiceSoap">
            <soap:address location="http://MigrationAuthority/MigrationService.asmx"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

The following sections provide detail on the data transferred for the available service calls.

7.1 MigrationServiceInformation

Invoking this call returns human readable information describing the services available. Appropriate information includes MA URL, MA Practice Statement, and a human readable description of the service.

7.2 GetMAKey

This call requests the MA to download its public key. The response must be in the form of a certificate.

7.3 Upload

This call sends the Migration Package to the MA. The call also includes a flag to instruct the MA whether the Migration Package is intended for vaulted or transitory status, and a target for the migration package (if transitory). The target can be either of a target public key, a certificate, a URI, or other (implementation specific). Upon successful upload, the MA returns an upload result and the ID of the Migration Package.

7.4 Download

This call sends a Migration Package ID, the public key of the parent for the downloaded key, optionally attestation information for the parent, and optionally a key to the MA in a request for download of the Migration Package. The attestation information can prove the parent is on the desired TPM and of an appropriate type (non-migratable or CMK as appropriate). The key provides confidentiality of the Migration Package if desired. The download result is the desired Migration Package.

7.5 Query

This call requests a list (number = record count) of available Migration Packages. The result is a list of available Migration Packages with the following information: ID, Description, Upload Time, Data Size, Key UUID, Owner Name, Parent UUID, and list of certificate hashes.

7.6 QueryByXPath

This call is the same as Query except it requests a list of a subset of the available keys. The requesting platform provides an XML Root Node name. The result is a list of available Migration Packages with the following information: ID, Description, Upload Time, Data Size, Key UUID, Owner Name, Parent UUID, and list of certificate hashes.

7.7 Delete

This call requests the MA to delete a list of Migration Packages identified by UUID.

8 Interoperability

8.1 TSS

8.1.1 Key Registration

It is highly desirable that users of TCG compliant platforms use the key storage faculties of the TSS and register the keys with the TSS in order to facilitate the identification of TPM keys and the construction of Migration Packages in a standard fashion. If a vendor chooses not to register keys within the TSS, they are required to provide the necessary client service to generate Migration Packages conformant to this specification in order to ensure proper business continuity capabilities for customers.