To: IEEE P1619.3 Task Group
From: P1619.3 [subgroup or ad hoc committee], Members:
[Subhash Sankuratripati, NetApp]
[Ravi Kavuri, NetApp]
[Gaurav Agarwal, NetApp]
[Scott Kipp, Brocade]
[Landon Noll, Cisco]
[Kevin Marks, Dell]
[Jon Hass, Dell]
[Larry Hofer, Emulex]
[Glen Jaquette, IBM]
[Walt Hubis, LSI]
[Matthew Ball, MV Ball Consulting]
[Robert A. (Bob) Lockhart, nCipher]
[Jon Holdman, Sun]
[Luther Martin, Voltage Security]
[Michael Marcil, Vormetric]

Deleted: e

Date: July 9, 2008

Deleted: July 1, 2008

Deleted: June 26, 2008

Purpose: Proposed changes against P1619.3/D3 to incorporate [Add Sections 4, 5 & 6 into the draft._]
[NOTE: Draft number proposed against should replace red x]

Introduction

The P1619.3 [subgroup or ad hoc committee] has been working on a proposal to create [Fill in an overview of what the committee is proposing].

[Please delete anything between brackets (including the brackets) and replace with the appropriate proposals]

[Rules and Guidance

a) *Diagrams can be submitted in color or grayscale. You may be asked to convert it if time is not on the editor's side at the moment.*

b) *All* [black bracketed] *text should be replaced with the appropriate information.*

i) *Note please delete this bullet completely. The format of the text is there for example purposes.*

c) *All text in dark red italics should be cut out of a document prior to submission (includes this entire section with numbering).*

d) *If you have verbiage or information that belongs in a section that Bob L. did not include you as creating, ignore Bob L.(this once only) and create away!*

e) *All dates that must be updated are automatically updated as you open & save the document. You should replace them with fixed dates for proposal and tracking purposes.*

f) *If you are not running a PC with Windows using Word, do not enable the macros. You should also avoid deleting them. Any documents that have to be cut and pasted back into a good macro document will cost you $25 to be put towards the WTSS subgroup (see P1619.3 meeting minutes dated September 17 2007).*

g) *Bob Lockhart will be participating off and on in all subgroups to monitor progress and assist with the documents as needed.]*

Changes to P1619.3/D3

[Change the red x to the appropriate draft number]

[Note any sections that are to be completely removed from the existing document.]
[Note sections that contain changes if the section is not to be fully deleted.]

1. Normative References

[Include any normative references in this section]

2. Definitions, acronyms, and abbreviations

For the purposes of this proposal, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards, Seventh Edition,* should be referenced for terms not defined in this clause.

[Ensure that definitions, acronyms and abbreviations do not already exist in the above reference]

2.1 Definitions

[2.1 i)term1: *select the 'definitions' and select 'format terms and definitions' in the IEEEStdsTemplate tool bar. If you do not have macro enabled please enter the number manually.]*

Deleted: *3.1*

2.2 Acronyms and abbreviations

[LAH List Acronyms (and/or abbreviations) Here]

[I have to manually edit the numbers here so just use standard body text formatting.]

3. General Overview

[Place any overview information as it pertains to the proposal in this section. Use section numbers appropriately. The editor reserves the right to move information from any section to a more appropriate section based on workgroup feedback]
[Architecture and Name Space belong in this section]
[We may need to move some or all of Name Space to section 5 or give it a separate section]

4. Key Management Objects

This section describes KM objects as they are transmitted across the wire to a KM client. Attributes that are 'persistent' across clients are listed in '**bold font'**. Attributes that are 'optional' are listed in '*italicized font*'.

4.1 Key

Scope: Client & Server.

The Key object consists of the key blob (potentially wrapped) along its meta-data.

4.1.1 Attributes

A key object distributed by a KMS contains the following attributes:

- **KEY_ID** (Type: SO_GUID)
- **FRIENDLY_NAME (**Optional: Type:String) Not necessarily unique within a KMS as additional attributes may be used to make a unique reference.

  Note: It should be possible to request a key by its Friendly_name (plus additional reference attributes if needed), and this may be used to hold prior key names for legacy key applications.

- **STATE** (Type:String) EDITORIAL: Reference back to the relevant section.
- T_EXPIRED (Type: UTC - time beyond which the key should not be used to encrypt new data)
- T_DISABLED (Type: UTC - time beyond which the key should be used)
- T_CACHED (Type: 64-bits – seconds that the key may be cached for. This may be differentiated by endpoint)
- **CIPHER_TYPE** (Type:String OID – **TODO:** Insert)
- KEY_BLOB (Object as defined in the next section) (This may be differentiated by endpoint, and is constructed from an immutable key value, the storage and representationof which is outside of this standard)
- *VENDOR_SPECIFIC_EXTENSIONS*
- *APPLICATION_EXTENSIONS*
- *CACHING_POLICY*
- (VERSIONING_INFORMATION:)

  Narrative: A KMS shall represent versioning of Key objects using two values: Version, which is an incrementally increasing value, representing changes to Key attributes, including changes to policies referenced by the key, and a GMT dateTime value representing the time of the last change. KMS_Clients may treat the combination as an opaque token, or use the values to protect against updates of stale copies. KMS servers may construct these values customized to the requestor, or maintain them globally independent of the endpoints. (for example, if a policy for a key changes, but that change is not relevant for an endpoint, the KMS may or may not represent update the versioning_Information. Versioning information will not change due to auditing activity, reference, inclusive Realm_Associations or backup.

  - VERSION (Type: unsigned Numeric)

    (NOTE: It must be possible for an endpoint to request key object if altered since a reference version)

  - EDIT_DATETIME (TYPE: DATETIME)

    (NOTE: It must be possible for an endpoint to request a list of key objects altered since a reference time)

Comment [MM1]: put this note into referenced section

Deleted: <#>*PEP_ID (Type:SO_GUID)*¶

In addition, a KMS must be capable of representing the following attributes and references:

- REALM_ASSOCIATIONS
  (Note: keys will not be delivered to endpoints without compatable Realm rights)
- *WRAPPING_POLICY*
- *DESCRIPTION* (Type:String)
- SOURCE
  Represents the identity of the originator of the key value. (a specificKMS, a specific client, a specific PEP)
- *ATTRIBUTE_ASSOCIATIONS*
  (A list of named value pairs useable as an alternate mechanism to define or reference a key . Keys can be retrieved by their key_ID or alternatively by an ANDED match on these NVPs)

Comment [MM2]: open for discussion.

In addition, a KMS must be capable of representing the following associations:

- *USE_BY_CLIENTS*
  (Note: does not alterVersioning_Information for a key itself. Note since any given key may be used by multiple clients or PEPs, this tracking must be maintained, so the KMS is capable of initiating unsolicited updates to a KMS_Client when a key's attributes or policies change)
- *USE_BY_PEPS (see above note)*

4.1.2 States

As defined in the key state diagram (Editorial: as defined by the architecture sub-committee)

4.1.3 Operations

- Create
- Get
- Store

4.2 Key Blob

4.2.1 Attributes

— ProtocolVersion (Type:int and defined as 1 for this version of the standard – This attribute will be remain constant for all clients for this version of the standard.).

— WRAPPING_TYPE (Type:String) – and can take any of the values as listed in the key wrapping section.

— Length (Type:int)

— Data (Type: Character Array)

Editorial: CMS will be used to wrap keys. The updates necessary to add key wrapping will be done at a later point.

4.3 Key_Template

Scope: Client & Server.

The Key_Template object consists of attributes and policies which may be inherited when creating a key (either by the KMS Admin, or by a key request). It does not represent any actual key.

It should be possible to make a key creation request "byTemplate", with or without additional dataset bindings. It is not possible to make a key retrieval request "byTemplate" without distinguishing dataset bindings.

Discussion: I suppose one could think of "template" as an additional "dataset binding" and use the same service as previously envisioned. The important notion here is the ability to predefine all the policy and attributes into some template to avoid having to manage all this separately.

4.3.1 Attributes

A Key_Template object defined within a KMS contains the following attributes:

- **KEY_TEMPLATE_ID** (Type: SO_GUID)
- **FRIENDLY_NAME** (Optional: Type:String) Unique within a KMS
  Note: It should be possible to request a key creation by template using its Friendly_name
- **CIPHER_TYPE** (Type:String OID – **TODO:** Insert)
- *VENDOR_SPECIFIC_EXTENSIONS*
- *APPLICATION_EXTENSIONS*
- *CACHING_POLICY*
- (VERSIONING_INFORMATION:)
  - VERSION (Type: unsigned Numeric)
  - EDIT_DATETIME (TYPE: DATETIME)
- REALM_ASSOCIATIONS
- *WRAPPING_POLICY*
- *DESCRIPTION* (Type:String)

4.4 ENDPOINT_TYPE

Endpoint_Type is an object to simplify the need to exchange capabilities between a KMS_CLIENT or PEP and a KMS_SERVER, as well as managing a collection of capabilities at the Server. (Discussion point: It would be desirable if these values were standardized in some registry.) An Endpoint_Type will always equate to a deterministic set of capabilities, though the converse need not be true. During registration, KMS_Clients or PEPs will present identifying information that will allow a KMS_Server to map it to an Endpoint_Type.

4.4.1 Attributes

- ENDPOINT_TYPE_ID (Type:TBD)
- CAPABILITIES (Note: common registry should allow retrieval of such characteristics as has-certificate, understands-time, min and max keyID lengths, never-exposes-key, hasHSM, etc.)

4.5 REALM (optional)

Realms are used to segment objects into separate administrative domains.

Administrative users and endpoints requesting key services will have “RealmAssociations” which will allow many to many representations specifying differing rights. For example, a policy may be deleted by an administrator belonging to a realm which also has delete capabilities on the policy, while an administrator in a realm with only read rights may use the policy, but can not delete or edit it. While a KMS may implement administrative “Roles”, Realms allow segmentation based on data characteristics rather than functional capabilities.

When a KMS provides Realm support, the KMS must insure no object is assessable unless the requesting endpoint or administrator has been properly associated with an incorporating realm that allows the access. Realms must allow the following distinctions: create, edit, delete, read, reference (use but not view). The most privileged right from any associated realm may be use to determine an access.

4.5.1 Attributes

- REALM_ID (Type:Realm://domain/realm-name where domain is a string that is in compliance with DNS name as defined by RFC 1034.
- DESCRIPTION

Comment [MM3]: or other format. Discuss best way to represent.

4.6 PEP (Policy Enforcement Point / Cryptographic Unit)

Scope: Client & Server

Narrative: While there is no direct communication with a PEP, there will be certain end points which may want to present an identity to the KM Server, so key information can be conveyed in a secure and predictable manner to the CU.

4.6.1 Attributes

Deleted: ¶

- PEP_ID (Type:SO_GUID)
- Array of Name, Value pairs.

NOTE: The name of the attributes that are part of the PEP object shall follow the following convention:

pep://domain/context/attribute-name where domain is a string that is in compliance with DNS name as defined by RFC 1034.

In addition – the following domin/context combination – ieee.org/siswg/ is reserved for use by this standard.

- REALM_ASSOCIATIONS (Server Only)
- ENDPOINT_TYPE_ID (TYPE:TBD Server Only.)

  DiscussionPoint: Prefer this to be some IEEE registry. note: KMS_Clients shall provide a CIM_PhysicalElement or CIM_SoftwareIdentity object upon registration of a PEP, which will allow a KMS to map an entity to a TYPE_ID. Or perhaps we define a new CIM object derived from common ones to include capabilities we want that may not be determined by attributes in the mentioned CIM objects.

  Comment [MM4]: matt prefers mechanism to avoid a registry.
- CLIENT_ASSOCIATIONS (Server Only)

  Note this might be done with a Client_Group. A PEP will initially be associated with full rights granted to its registering Client. Other behaviors to manage associations is outside the scope of this spec., but a KMS must be capable of conforming to a policy or configuration that prevents a PEP from receiving its key from an "unauthorized" client. This can easily be accomplished by restricting access to keys both to authorized KMS clients and authorized PEPs.)
- *AUTHENTICATION_POLICY*
- *AUTHENTICATION_VALUES (It must be possible for a PEP to authenticate to the KMS through an untrusted KMS_CLIENT)*
- *List of {CREDENTIAL_LENGTH,CREDENTIAL_VALUE} tuples.*
- *WRAPPING_POLICY. (Note: this may typically be inherited via endpoint_type_id)*
- *WRAPPING_VALUES (Since PEPs can uniquely protect some wrapping values, such as private keys, data can pass through a KMS_Client without exposure)*

4.7 Client

Scope: Client & Server.

The client object consists of its credentials and capabilities.

4.7.1 Attributes

- CREDENTIAL_TYPE (Session, Username/Password, Symmetric / Asymmetric key, SSO, CHAP)
- List of {CREDENTIAL_LENGTH,CREDENTIAL_VALUE} tuples.
- REALM_ASSOCIATIONS
- **ENDPOINT_TYPE_ID**
- WRAPPING_POLICY

4.7.2 States

- Active
- Disabled/Locked
- Authenticated

4.7.3 Operations

- Create
- Delete
- Authenticate
- Disable

NOTE: All of these operations with the exception of authenticate are performed by way of KM Console operations and are outside the scope of the standard.

4.8 Capability

4.8.1 Attributes

— Name (Type: String)

4.8.2 States

None.

4.8.3 Operations

No direct operations. The capability object is sent as part of the capability negotiation operation, or constructible by reference to an endpoint type.

Deleted: .

1

Deleted: Key Manager¶
Scope: Server only.¶
NOTE: In this version of the specification, a key manager is the same as a client as there no operations that are KM ⇔ KM specific.

4.9 Data Sets

Deleted:

Scope: Client, PEP & Server.

Data sets represent manageable units of encrypted data. Data sets are expressed as selection rules that can be applied to data set attributes such as file path, tape volume id, server IP, or a range of disk blocks. There should be flexibility in defining what a data set is, depending on the position of the encryption agent "in the stack" of the storage infrastructure.

Once the data sets are identified, keys may be associated to data sets via a key assignment policy.

4.9.1 Attributes

- NAME
- VALUE
- SIZEOF_VALUE

4.10 Client Groups

Scope: Server only.

Clients may be grouped together for ease of management. This grouping may be static – i.e. clients are explicitly added into a group or dynamic i.e. based on a regular expression match on client attributes.

4.10.1 Attributes

- TYPE (STATIC or DYNAMIC)
- List of CLIENT_SO_GUIDs (only in case of static binding)
- List of {PATTERN, ATTRIBUTE} tuple.
- REALM_ASSOCIATIONS

4.11 Key Groups

Scope: Server only.

Keys may be grouped together for ease of management. This grouping may be static – i.e. explicitly added into a group or dynamic i.e. based on a regular expression match on dataset attributes.

4.11.1 Attributes

- TYPE (STATIC or DYNAMIC)
- List of CLIENT_SO_GUIDs (only in case of static binding)
- List of {PATTERN, DATASET} tuple.
- REALM_ASSOCIATIONS

5. Key Management Policies

A policy is a deliberate plan of action to guide decisions and achieve rational outcome(s). In the same vein, Key Management Policies are used to guide assignment, retention, wrapping, replication & access control decisions on keys.

The scope of some policy objects will extend to an endpoint.

5.1 Key Assignment Policy

Key Assignment Policies contain logic that is able to determine which data set should be encrypted with which key using which algorithm (e.g. encrypt and sign all emails sent outside of the company. Sign all tapes with a unique key per tape, etc.)

A key assignment policy determines

- the type of unencrypted data that is determined to be encrypted with a specific key.
- how often to generate new keys

Therefore, the key assignment policy encapsulates both key generation and key scope policies. This is done to fit regular usage patterns. For example, when a tape is loaded into a drive, the drive will request a key by data, and will receive both a key that may be used on that drive, as well as a policy notifying the drive whether all tapes should be encrypted with this key, or only the current tape.

The key assignment policy is determined by the set of supported data set attributes, and is encoded as a set of name, value pairs.

The policy is interpreted to mean that the key may be used whenever all the named parameters have values equal to the values of the data presented to the client. So, for example, in the following encryption policy:

container attribute name = ”storage_server_name” value=”Ireland.com”

data attribute name = ”financial”

The provided key may be used to encrypt all of the data tagged as “financial” on the storage server named “Ireland.com” with the key listed in the KeyExchangeStructure.

Note that there is a difference between a data set binding and an assignment policy, and the KMS must track both. An organization may set a policy to encrypt a pool with a specific key, but due to key rotation policies, not all tapes within the pool will have been encrypted with that key. Therefore, assignment policies specify the current desired behavior, whereas the KMS will store all data set bindings that are reported to it, for future audit and key query commands. State logic within the KMS may allow for the automatic creation of key assignment policies based on the “most recent” data set binding.

5.1.1 Attributes

- KEY_ASSIGNMENT_POLICY_ID
- DESCRIPTION
- REALM_ASSOCIATIONS

Deleted: (Source: Wikipedia)

Deleted: all

Deleted: are ‘Server only’.

Deleted: ¶

5.2 Retention Policy

5.2.1 Overview

The retention policy dictates the duration for which protected data, hence the key with which it is encrypted, is accessible to a given client. It also dictates when new data should no longer be encrypted with a given key.

This policy should be superseded by the key life cycle.

5.2.2 Attributes

- RETENTION_POLICY_ID
- DESCRIPTION
- REALM_ASSOCIATIONS
- T_EXPIRATION
- T_DISABLE

5.2.3 States

- Created
- Assigned
- Enforcing
- Disabled

5.2.4 Operations

- Add
- Associate
- Disassociate
- Delete

5.3 Wrapping Policy

The wrapping policy indicates whether a key should be wrapped prior to being dispatched to a client. This policy may be referenced from the key object, ?a key group?, a client object, ?a ClientGroup?, a PEP object, or a Endpoint_type. If multiple policies are defined then the order of precedence shall be per the following:

1. Key shall prevail over KeyGroup
2. Key or KeyGroup shall prevail over PEP
3. PEP shall prevail over Pep's Endpoint_Type
4. Client shall prevail over Client's Endpoint_Type
5. If both a PEP and Client specify (or inherit) a Wrapping Policy, the key will be double wrapped, first by policy prevailing for the PEP, then by prevailing ClientPolicy. The KMS_Client will unwrap the key and pass the wrapped Client_Key for subsequent unwrapping.

Deleted: If a key has a wrapping policy, then it SHOULD override the client's wrapping policy.

Deleted: applied either at

Deleted: level

Deleted: or at

Deleted: level

5.3.1 Attributes

Comment [MM5]: Subhash, this was an incomplete sentence

Deleted: <#>¶
INFORMATIVE: This policy when applied at the key level can dictate whether ¶

Deleted: <#>*CU_ID (when enforced at a key level, then this allows the key to be locked down to a particular client).*¶

- WRAPPING_TYPE (asymmetric / symmetric / signature)
- WRAPPING_MODE (as listed in the key blob section)

5.3.2 States

- Created
- Assigned
- Enforcing
- Disabled

5.3.3 Operations

- Add
- Associate
- Disassociate
- Delete

5.4 Audit Policy

Audit policies state the auditing requirements that need to be enforced on keys and clients.

Deleted: ory

TODO: Bob Lockhart to provide new content.

5.4.1 Attributes

— Operation type
— Event type (to trigger, Log, SNMP trap, etc.) Mandatory: Log
— Event Dispatch Destination (Local, SNMP, syslog) Mandatory: Local, syslog.

NOTE: The only mandatory type that should be supported is ‘log’ and the mandatory dispatch destinations are local and syslog.

5.4.2 States

— Created
— Assigned
— Enforcing
— Disabled

5.4.3 Operations

— Add
— Associate
— Disassociate
— Delete

5.5 Access/Distribution Policy

Key access policies encode which clients and key management servers may access which keys. This may be controlled by the clients or PEPs, as a key creation request may set the data set bindings of a key, but it is enforced by the KMS, which lookup tables storing keys to data assignments, and clients to data permissions, always enforcing any realm restrictions.

In addition, the KMS administrator for a key's realm may alter a key's access and distribution policy.

This policy is referenced by a key or key group.

Note: this policy governs what endpoints MAY receive a key, but can not be used to determine which endpoints in fact received any given key.

5.5.1 Attributes

- SO_GUID
- Client or clientGroup list
- PEP list
- KM server list.
- REALM_ASSOCIATIONS

5.5.2 States

- Created
- Assigned
- Enforcing
- Disabled

5.5.3 Operations

- Add
- Associate
- Disassociate
- Delete

Deleted: Replication Policy¶
The key replication policy describes the set of key management servers the keys should be replicated to. The synchronization protocol is outside the scope of this version of the standard since KMSS operations are out of scope. ¶
It is assumed that a KM server would act as a 'trusted client' and utilize KMCS operations to synchronize keys.¶
<#>Attributes¶
<#>List of KM servers¶
<#>States¶
<#>Created¶
<#>Assigned¶
<#>Enforcing¶
<#>Disabled¶
<#>Operations¶
<#>Add¶
<#>Associate¶
<#>Disassociate¶
<#>Delete¶
¶
¶
Page Break

Deleted: is implicitly

Deleted:

Deleted: the clients

Deleted: s

Deleted: .

Deleted: applied

Deleted: at

Deleted: /

Deleted: level

Deleted: ¶

5.6 Caching Policy

The key caching policy dictates whether a key shall be cached by a KM client and if so, the duration for which it can.

Attributes

— CACHING_TYPE, T_CACHE_INTERVAL tuples (list)

Note: It should be possible to allow different time intervals for caching depending on the security of the caching along different attributes such as HSM, TPM, neverExposedHardware,

5.6.1 States

— Created
— Assigned
— Enforcing
— Disabled

5.6.2 Operations

— Add
— Associate
— Disassociate
— Delete

6. Key Management Operations

6.1 Register Endpoint

Scope: KMCS Ops, including registration for KMS_Client or PEP

Editorial comment We need to fill this operation out. e.g. Identify who is registering, type and/or capabilities, how authenticate, certs, etc. send cim object

6.2 Authenticate

Scope: KMCS

6.2.1 Overview

A client needs to authenticate with the KM server to perform any sensitive operations. Authentication is accomplished either at the transport level (SSL/TLS) or at the object/messaging level. Every request shall contain the "credential" object so that the KM server can validate the client.

6.2.2 Input / Output / Error

- (I): Client
- (O): Credentials (If the request type is login and not validation)
- (E): E_INVALID_CREDENTIALS
- (E): E_UNSUPPORTED_AUTHENTICATION_MODE

6.3 Capability Negotiation

Scope: KMCS

6.3.1 Overview

The client sends its capabilities to the server and the server returns back a list of capabilities it supports. If none of the capabilities are supported, then it returns back an empty list.

6.3.2 Input / Output / Error

- (I): Client
- (I): List of Capability Objects
- (O): List of Capability Objects that are supported by the KM server

6.4 Get Server Capabilities

- (I): Client
- (O): List of Capability Objects.

6.5 Create/Generate Key

Comment [MM6]: note that any object can belong to multiple realms. it is a many to many relationship. Template policies are for inheritance and can be overridden by an object specific policy.

Scope: KMCS, KM Console

6.5.1 Overview

A client upon authentication invokes the Generate key operation to generate a new key by passing in the KeyTemplateID and/or DataSet context in which this key would be used so that the KM can apply the appropriate policies.

Deleted: g

6.5.2 Input / Output / Error

- (I): Client
- (I) *PEP_ID* or EndPoint's CIM Object - Identifier of final destination for the key.
- (I): List of Dataset objects.
- (O): Key (including unique So_Guid)
- (E): …

6.6 Store Key

Scope: KMCS, KM Console

6.6.1 Overview

Keys that are generated at the client can be stored in the KM server by invoking its store functionality.

6.6.2 Input / Output / Error

(I): Client

(I): List of Dataset objects.

(I) *PEP_ID* or EndPoint's CIM Object - Identifier of final destination for the key.

(O): Key_SO_GUID

Deleted: ¶

(I) *Friendly_Name*

(E)…

6.7 Get Key

6.7.1 Overview

Clients invoke the get key operation to fetch keys from the KM server. They may invoke the query based on either a Key ID or FriendlyName, and/or based on the Dataset attributes. When querying based on dataset attributes, the KM returns a key based on the application template and the policies that govern the key and the client.

Deleted: DataSet

Deleted: .

6.7.2 Input / Output / Error

(I): Client

(I) PEP

(I): List of Dataset objects.

(O): Key

(E): …

6.8 Push Audit Message

6.8.1 Overview

This operation is intended to be used by 'super' clients that maintain local caches of keys and ship them out to cryptographic units on demand. This will ensure that the KM Server can be a central audit repository for any/all accesses to keys.

Discussion: Marcil: I think this should be broader than presented, which only covers one type of message. For example, an encrypting drive may be configured locally to unlock the drive and therefore no longer need a key. Or an existing key may be used for another device. It would be good to have an audit message that reflects these. Audit messages should also be batchable and uploaded in a file.

6.8.2 Input / Output / Error

(I): Client or PEP

(I): Key_SO_GUID

(I): Message (Optional)

(O): Boolean – SUCCESS/FAILURE.

(E): …

6.9 Get Random Bytes

6.9.1 Input / Output / Error

- (I): Client (question: why?)
- (I): numbers of bytes desired
- (O) Base64 Encoded bytes

6.10 GetStatus --KMS_Client Service (optional) -- [Server initiated]

Comment [MM7]: this is a contentious point. Worthy of further discussion.

Server asking client for status.

Discussion: To aid KMS Administrators to provide key management for endpoints, we will need some mechanism to gather status on endpoints, including operating mode, Keys in use, status of any rekeying, We could try and define some predetermined status types or just allow these to come back with what the KMS Client and PEP can provide.

Discussion: This probably matches the concepts of some existing WSMAN service.

- (I): Target of Interest Type: filter expression
- (I) *Locale* – requested language for any NVPs
- (O) *Locale – language used for NVPs*
- (O) Endpoint + NVPs

6.11 UpdatePending --KMS_Client Service (optional) [Server initiated]

Server notifying client of relevant updates.

KMS_Clients expose this service. KMS Servers will repeat this service until the client acknowledges currency by virtue of matching UpdateVersioningTokens. Note, these tokens are used for sequencing between this service and GetChangeList. A simple implementation of this would be for the KMS Server to maintain a VersioningToken to represent the latest version of "Everything" for a KMS_Client or PEP and a response to a GetUpdateList that returns everything.

- (I): Type (Keys, AllKeys and custom types)
- (I) Scope: (KMS_Client or PEP identifier)
- (I) UpdateVersioningTokens - KMS server sends its tokens representing the state to which the client (or PEP) needs to update.
- (O) UpdateVersioningTokens - KMS client sends its tokens representing the state it has received

6.12 GetUpdateList --KMS_Server Service [Client initiated]

Discussion: can this be done with a WS-ENUMERATE service?

- (I): Type: (Keys, AllKeys and custom types)
- (I) Scope: (KMS_Client or PEP identifier)
- (I) UpdateVersioning Tokens (zero values will result in all requested instances of requested type)
- (O) requested objects
- (O) New UpdateVersioningTokens

7. Key Management Transport

[Supported transport protocols go here]

8. Key Management Messaging

[This is an additional section that I am proposing we add to help complete the standard. It does not currently exist in Draft 1.]
[This section would contain normative information for the XML and/or TLV formats we decide to support]

Annex A

(informative)

Bibliography

[List all bibliographic material here]

Annex B (informative)

Example Use Cases

[Objects & operations or use cases should add the appropriate use cases here]

Annex C (informative)

XML and TLV Schema Definitions

C.1 XML Schema

[Additional messaging group information for selected XML syntax goes here]

C.2 TLV Schema

[Additional messaging group information for selected TLV schema goes here]