This is a Non-Standards Track Work Product. The patent provisions of the OASIS IPR Policy do not apply.

# Key Management Interoperability Protocol Usage Guide Version 1.2

## Committee Note 01

## 11 November 2014


### Specification URIs

**This version:**

http://docs.oasis-open.org/kmip/ug/v1.2/cn01/kmip-ug-v1.2-cn01.doc (Authoritative)
http://docs.oasis-open.org/kmip/ug/v1.2/cn01/kmip-ug-v1.2-cn01.html
http://docs.oasis-open.org/kmip/ug/v1.2/cn01/kmip-ug-v1.2-cn01.pdf

**Previous version:**

http://docs.oasis-open.org/kmip/ug/v1.2/cnprd01/kmip-ug-v1.2-cnprd01.doc (Authoritative)
http://docs.oasis-open.org/kmip/ug/v1.2/cnprd01/kmip-ug-v1.2-cnprd01.html
http://docs.oasis-open.org/kmip/ug/v1.2/cnprd01/kmip-ug-v1.2-cnprd01.pdf

**Latest version:**

http://docs.oasis-open.org/kmip/ug/v1.2/kmip-ug-v1.2.doc (Authoritative)
http://docs.oasis-open.org/kmip/ug/v1.2/kmip-ug-v1.2.html
http://docs.oasis-open.org/kmip/ug/v1.2/kmip-ug-v1.2.pdf

**Technical Committee:**

OASIS Key Management Interoperability Protocol (KMIP) TC



**Chairs:**

Saikat Saha (Saikat.saha@oracle.com), Oracle
Tony Cox (tjc@cryptsoft.com), Cryptsoft Pty Ltd.

**Editors:**

Indra Fitzgerald (indra.fitzgerald@hp.com), HP
Judith Furlong (Judith.Furlong@emc.com), EMC Corporation


**Related work:**

This document replaces or supersedes:

- *Key Management Interoperability Protocol Usage Guide Version 1.1*. Edited by Indra Fitzgerald and Robert Griffin. Latest version. http://docs.oasis-open.org/kmip/ug/v1.1/kmip-ug-v1.1.html.

This document is related to:

- *Key Management Interoperability Protocol Specification Version 1.2*. Edited by Kiran Thota and Kelley Burgin. Latest version: http://docs.oasis-open.org/kmip/spec/v1.2/kmip-spec-v1.2.html.
- *Key Management Interoperability Protocol Profiles Version 1.2*. Edited by Tim Hudson and Robert Lockhart. Latest version: http://docs.oasis-open.org/kmip/profiles/v1.2/kmip-profiles-v1.2.html.
- *Key Management Interoperability Protocol Test Cases Version 1.2*. Edited by Tim Hudson and Faisal Faruqui. Latest version: http://docs.oasis-open.org/kmip/testcases/v1.2/kmip-testcases-v1.2.html.

## Abstract:

This document is intended to complement the Key Management Interoperability Protocol Specification by providing guidance on how to implement KMIP most effectively to ensure interoperability and to address key management usage scenarios.

KMIP v1.2 enhances the KMIP v1.1 standard (established in February 2013) by

1) defining new functionality in the protocol to improve interoperability;
2) defining additional Test Cases for verifying and validating the new functionality;
3) providing additional information in the KMIP Usage Guide to assist in effective implementation of KMIP in key management clients and servers; and
4) defining new profiles for establishing KMIP-compliant implementations.

The Key Management Interoperability Protocol (KMIP) is a single, comprehensive protocol for communication between clients that request any of a wide range of encryption keys and servers that store and manage those keys. By replacing redundant, incompatible key management protocols, KMIP provides better data security while at the same time reducing expenditures on multiple products.

## Status:

This document was last revised or approved by the OASIS Key Management Interoperability Protocol (KMIP) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=kmip#technical.

Technical Committee members should send comments on this document to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/kmip/.

## Citation format:

When referencing this document the following citation format should be used:

**[kmip-ug-v1.2]**

*Key Management Interoperability Protocol Usage Guide Version 1.2.* Edited by Indra Fitzgerald and Judith Furlong. 11 November 2014. OASIS Committee Note 01. http://docs.oasis-open.org/kmip/ug/v1.2/cn01/kmip-ug-v1.2-cn01.html. Latest version: http://docs.oasis-open.org/kmip/ug/v1.2/kmip-ug-v1.2.html.

  

# Table of Contents



  

# 1 Introduction

This Key Management Interoperability Protocol Usage Guide Version 1.2 is intended to complement the Key Management Interoperability Protocol Specification **[KMIP-Spec]** by providing guidance on how to implement the Key Management Interoperability Protocol (KMIP) most effectively to ensure interoperability and to address key management usage scenarios. In particular, it includes the following guidance:

- Clarification of assumptions and requirements that drive or influence the design of KMIP and the implementation of KMIP-compliant key management.
- Specific recommendations for implementation of particular KMIP functionality.
- Clarification of mandatory and optional capabilities for conformant implementations.

Descriptions of how to use KMIP functionality to address specific key management usage scenarios or to solve key management related issues. A selected set of conformance profiles and authentication suites are defined in the KMIP Profiles specification **[KMIP-Prof].**

Further assistance for implementing KMIP is provided by the KMIP Test Cases document **[KMIP-TC]** that describes a set of recommended test cases and provides the TTLV (Tag/Type/Length/Value) format for the message exchanges defined by those test cases.

## 1.1 References (normative)


**[ECC-Brainpool]**
*ECC Brainpool Standard Curves and Curve Generation v. 1.0.19.10.2005*, http://www.ecc-brainpool.org/download/Domain-parameters.pdf.

**[FIPS 180-4]**
*Secure Hash Standard (SHS)*, FIPS PUB 180-4, March 2012,
http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

**[FIPS 186-4]**
*Digital Signature Standard (DSS)*. FIPS PUB 186-4. July 2013.
http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

**[FIPS 197]**
*Advanced Encryption Standard (AES)*. FIPS PUB 197. November 26, 2001.
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

**[FIPS 198-1]**
*The Keyed-Hash Message Authentication Code (HMAC)*. FIPS PUB 198-1. July 2008.
http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf

**[KMIP-Spec]**

*Key Management Interoperability Protocol Specification Version 1.2.* Edited by Kiran Thota and Kelley Burgin. Latest version: http://docs.oasis-open.org/kmip/spec/v1.2/kmip-spec-v1.2.doc.

**[KMIP-Prof]**

*Key Management Interoperability Protocol Profiles Version 1.2*. Edited by Tim Hudson and Robert Lockhart. Latest version: http://docs.oasis-open.org/kmip/profiles/v1.2/kmip-profiles-v1.2.doc.

**[PKCS#1]**

RSA Laboratories. PKCS #1 v2.1: RSA Cryptography Standard. June 14, 2002.
ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf

**[PKCS#10]**

RSA Laboratories. PKCS #10 v1.7: Certification Request Syntax Standard. May 26, 2000.
ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-10/pkcs-10v1_7.pdf

**[RFC1321]**

R. Rivest, *The MD5 Message-Digest Algorithm*, IETF RFC 1321, Apr 1992, http://www.ietf.org/rfc/rfc1321.txt

**[RFC1421]**

J. Linn, *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures*, IETF RFC 1421, Feb 1993, http://www.ietf.org/rfc/rfc1421.txt

**[RFC3647]**

S. Chokhani, W. Ford, R. Sabett, C. Merrill, and S. Wu. *RFC3647: Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework*. November 2003. http://www.ietf.org/rfc/rfc3647.txt

**[RFC4210]**

C. Adams, S. Farrell, T. Kause and T. Mononen, *Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)*, IETF RFC 2510, Sep 2005, http://www.ietf.org/rfc/rfc4210.txt

**[RFC4211]**

J. Schaad, *Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)*, IETF RFC 4211, Sep 2005**,** http://www.ietf.org/rfc/rfc4211.txt

**[RFC4949]**

R. Shirey. *RFC4949: Internet Security Glossary, Version 2*. August 2007. http://www.ietf.org/rfc/rfc4949.txt

**[RFC4880]**

J. Callas, L. Donnerhacke, H. Finney, D. Shaw and R. Thayer. *RFC4880: OpenPGP Message Format. November 2007*. http://www.ietf.org/rfc/rfc4880.txt

**[RFC5272]**

J. Schaad and M. Meyers, *Certificate Management over CMS (CMC)*, IETF RFC 5272, Jun 2008, http://www.ietf.org/rfc/rfc5272.txt

**[RFC5280]**

  

D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. *RFC5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* May 2008. http://www.ietf.org/rfc/rfc5280.txt

**[RFC5639]**

M. Lochter, J. Merkle, *Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation*, IETF RFC 5639, March 2010, http://www.ietf.org/rfc/rfc5639.txt.

**[SEC]**

SEC 2: Recommended Elliptic Curve Domain Parameters, http://www.secg.org/collateral/sec2_final.pdf.

**[SP800-38A]**

M. Dworkin. *Recommendation for Block Cipher Modes of Operation – Methods and Techniques*. NIST Special Publication 800-38A, Dec 2001. http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

**[SP800-38D]**

M. Dworkin. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. NIST Special Publication 800-38D. Nov 2007. http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf

**[SP800-56A]**

E. Barker, L. Chen, A. Roginsky, and M. Smid, *Recommendations for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography*, NIST Special Publication 800-56A Revision 2, May 2013, http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar2.pdf

**[SP800-57-1]**

E. Barker, W. Barker, W. Burr, W. Polk and M. Smid, *Recommendations for Key Management – Part 1: General (Revision 3)*, NIST Special Publication 800-57 Part 1 Revision 3, July 2012, http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf

**[SP800-67]**

W. Barker and E. Barker, *Recommendations for the Triple Data Encryption Algorithm (TDEA) Block Cipher*, NIST Special Publication 800-67 Revision 1, January 2012, http://csrc.nist.gov/publications/nistpubs/800-67-Rev1/SP-800-67-Rev1.pdf

**[X.509]**

International Telecommunications Union (ITU)-T, *X.509: Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks*, November 2008, https://www.itu.int/rec/T-REC-X.509/recommendation.asp?lang=en&parent=T-REC-X.509-200811-S

**[X9.31]**

ANSI, *X9.31: Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA)*. September 1998.

**[X9.42]**

ANSI, *X9.42: Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography*. 2003.

  

**[X9.62]**
ANSI, *X9.62: Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 2005.

**[**

## 1.2 References (non-normative)

**[KMIP-TC]**
*Key Management Interoperability Protocol Test Cases Version 1.2.* Edited by Tim Hudson and Faisal Faruqui. Latest version: http://docs.oasis-open.org/kmip/testcases/v1.2/kmip-testcases-v1.2.doc.

  

# 2 Assumptions

The section describes assumptions that underlie the KMIP protocol and the implementation of clients and servers that utilize the protocol.

## 2.1 Island of Trust

Clients may be provided key material by the server, but they only use that keying material for the purposes explicitly listed in the delivery payload. Clients that ignore these instructions and use the keys in ways not explicitly allowed by the server are non-compliant. There is no requirement for the key management system, however, to enforce this behavior.

## 2.2 Message Security

KMIP relies on the chosen authentication suite as specified in **[KMIP-Prof]** to authenticate the client and on the underlying transport protocol to provide confidentiality, integrity, message authentication and protection against replay attack. KMIP offers a wrapping mechanism for the Key Value that does not rely on the transport mechanism used for the messages; the wrapping mechanism is intended for importing or exporting managed cryptographic objects.

## 2.3 State-less Server

The protocol operates on the assumption that the server is state-less, which means that there is no concept of "sessions" inherent in the protocol. This does not mean that the server itself maintains no state, only that the protocol does not require this.

## 2.4 Extensible Protocol

The protocol provides for "private" or vendor-specific extensions, which allow for differentiation among vendor implementations. However, any objects, attributes and operations included in an implementation are always implemented as specified in **[KMIP-Spec],** regardless of whether they are optional or mandatory.

## 2.5 Server Policy

A server is expected to be conformant to KMIP and supports the conformance clauses as specified in **[KMIP-Spec].** However, a server may refuse a server-supported operation or client-settable attribute if disallowed by the server policy (whether expressed within or outside KMIP). Such a decision by the server may reflect the trust relationship with a particular client, performance impact of the requested operation, or any of a number of other considerations.

## 2.6 Support for Cryptographic Objects

The protocol supports key management system-related cryptographic objects. This list currently includes:

- Symmetric Keys

  

- Split (multi-part) Keys
- Asymmetric Key Pairs (Public and Private Keys)
- PGP Keys
- Certificates
- Secret Data
- Opaque (non-interpretable) cryptographic objects

## 2.7 Client-Server Message-based Model

The protocol operates primarily in a client-server, message-based model. This means that most protocol exchanges are initiated by a client sending a request message to a server, which then sends a response to the client. The protocol also provides optional mechanisms to allow for unsolicited notification of events to clients using the Notify operation, and unsolicited delivery of cryptographic objects to clients using the Put operation; that is, the protocol allows a "push" model, whereby the server initiates the protocol exchange with either a Notify or Put operation. These Notify or Put features are optionally supported by servers and clients. Clients may register in order to receive such events/notifications. Registration is implementation-specific and not described in the specification.

## 2.8 Synchronous and Asynchronous Operations

The protocol allows two modes of operation: synchronous and asynchronous. Synchronous (mandatory) operations are those in which a client sends a request and waits for a response from the server. Asynchronous operations (optional) are those in which the client sends a request, the server responds with a "pending" status, and the client polls the server for the completed response and completion status. Server implementations must support synchronous operations, but may choose not to support asynchronous operations.

## 2.9 Support for "Intelligent Clients" and "Key Using Devices"

The protocol supports intelligent clients, such as end-user workstations, which are capable of requesting all of the functions of KMIP. It also allows subsets of the protocol and possible alternate message representations in order to support less-capable devices, which only need a subset of the features of KMIP.

## 2.10 Batched Requests and Responses

The protocol contains a mechanism for sending batched requests and receiving the corresponding batched responses, to allow for higher throughput on operations that deal with a large number of entities, e. g., requesting dozens or hundreds of keys from a server at one time, and performing operations in a group. A Batch Error Continuation option is provided to indicate whether to undo all previous successful operations once a request in the batch fails; to continue processing requests after an earlier request in the batch fails; or to stop processing the remaining requests in the batch (but not undo previously successful operations). A special ID

  

Placeholder (see Section 3.18) is provided in KMIP to allow related requests in a batch to be pipelined.

## 2.11 Reliable Message Delivery

The reliable message delivery function is relegated to the transport protocol, and is not part of the key management protocol itself.

## 2.12 Large Responses

For requests that could result in large responses, a mechanism in the protocol allows a client to specify in a request the maximum allowed size of a response or in the case of the Locate operation the maximum number of items which should be returned. The server indicates in a response to such a request that the response would have been too large and, therefore, is not returned.

## 2.13 Key Life-cycle and Key State

**[KMIP-Spec]** describes the key life-cycle model, based on the **[SP800-57-1]** key state definitions, supported by the KMIP protocol. Particular implications of the key life-cycle model in terms of defining time-related attributes of objects are discussed in Section 3.5 below.

  

# 3 Using KMIP Functionality

This section provides guidance on using the functionality described in the Key Management Interoperability Protocol Specification.

## 3.1 Authentication

As discussed in **[KMIP-Spec]**, a conforming KMIP implementation establishes and maintains channel confidentiality and integrity, and provides assurance of server authenticity for KMIP messaging. Client authentication is performed according to the chosen KMIP authentication suite as specified in **[KMIP-Prof]**. Other mechanisms for client and server authentication are possible and optional for KMIP implementations.

KMIP implementations that support the KMIP-defined Credential Types or use other vendor-specific mechanisms for authentication may use the optional Authentication structure specified inside the Request Header to include additional identification information. Depending on the server's configuration, the server may interpret the identity of the requestor from the Credential structure, contained in the Authentication structure if it is not provided during the channel-level authentication. For example, in addition to performing mutual authentication during a TLS handshake, the client passes the Credential structure (e.g., a username and password) in the request. If the requestor's username is not specified inside the client certificate and is instead specified in the Credential structure, the server interprets the identity of the requestor from the Credential structure. This supports use cases where channel-level authentication authenticates a machine or service that is used by multiple users of the KMIP server. If the client provides the username of the requestor in both the client certificate and the Credential structure, the server verifies that the usernames are the same. If they differ, the authentication fails and the server returns an error. If no Credential structure is included in the request, the username of the requestor is expected to be provided inside the certificate. If no username is provided in the client certificate and no Credential structure is included in the request message, the server is expected to refuse authentication and return an error.

If authentication is unsuccessful, and it is possible to return an "authentication not successful" error, this error should be returned in preference to any other result status. This prevents status code probing by a client that is not able to authenticate.

Server decisions regarding which operations to reject if there is insufficiently strong authentication of the client are not specified in the protocol. However, see Section 3.2 for operations for which authentication and authorization are particularly important.

### 3.1.1 Credential

The Credential object defined in the **[KMIP-Spec]** is a structure used to convey information about the client, but the contents of this object are not managed by the key management server. The type of information convey within this object varies based on the type of credential.

  

KMIP 1.2 supports three credential types: *Username and Password*, *Device Credential* and *Attestation*.

#### 3.1.1.1 Username and Password Credential Type

**[KMIP-Spec]** defines the Username and Password structure for the Credential Type Username and Password. The structure consists of two fields: Username and Password. Password is a recommended, but optional, field, which may be excluded only if the client is authenticated using one of the authentication suites defined in **[KMIP-Prof]** For example, if the client performs client certificate authentication during the TLS handshake, and the Authentication structure is provided in the Message Request, the Password field is an optional field in the Username and Password structure of the Credential structure.

The Credential structure is used to provide additional identification information. As described above, for certain use cases, channel-level authentication may only authenticate a machine or service that is used by multiple clients of the KMIP server. The Credential structure may be used in this scenario to identify individual clients by specifying the username in the Username and Password structure.

#### 3.1.1.2 Device Credential Type

The Device Credential may be used to uniquely identify back-end devices by specifying Device as the Credential Type in the Credential structure.

The Device Credential may be used in a proxy environment where the proxy authenticates with the client certificate and supports KMIP while the back-end devices may not support KMIP or TLS. An example is illustrated below:

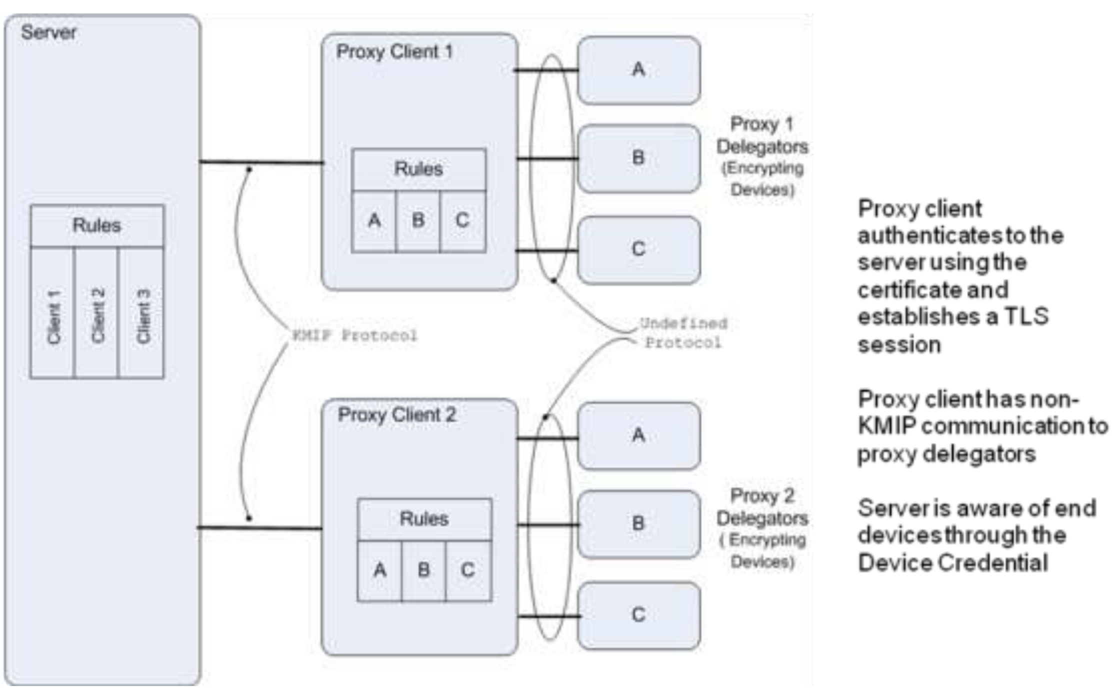

FIGURE 1: AGGREGATOR CLIENT EXAMPLE

The end device identifies itself with a device unique set of identifier values that include the device hardware serial number, the network identifier, the machine identifier, or the media identifier. For many of the self-encrypting devices there is a unique serial number assigned to the device during manufacturing. The ability to use network, machine, or media identifier explicitly should map to different device types and achieve better interoperability since different types of identifier values are explicitly enumerated. The device identifier is included for more generic usage. An optional password or shared secret may be used to further authenticate the device.

Server implementations may choose to enforce rules for uniqueness for different types of identifier values, combinations of TLS certificate used in combination with the Device Credential, and optionally enforce the use of a Device Credential password.

Four identifiers are optionally provided but are unique in aggregate:

1. Serial Number, for example the hardware serial number of the device
2. Network Identifier, for example the MAC address for Ethernet connected devices
3. Machine Identifier, for example the client aggregator identifier, such as a tape library aggregating tape drives
4. Media Identifier, for example the volume identifier used for a tape cartridge

The device identifier by choice of server policy may or may not be used in conjunction with the above identifiers to insure uniqueness.

These additional identifiers are generally useful for auditing and monitoring encryption and could according to server policy be logged or used in server implementation specific validation.

A specific example for self-encrypting tape drive and tape library would be:

1. the tape drive has a serial number that is unique for that manufacturer and the vendor has procedures for maintaining and tracking serial number usage
2. a password optionally is created and stored either on the drive or the library to help authenticate the drive
3. the tape drives may be connected via fiber channel to the library and therefore have a World Wide Name assigned
4. a machine identifier can be used to identify the tape library that is aggregating the device in question
5. the media identifier helps identify the individual media such as a tape cartridge for proof of encryption reporting

Another example using self-encrypting disk drives inside of a server would be:

1. the disk drive has a unique serial number
2. a password may be supplied by configuration of the drive or the server where the drive is located

  

3. the network identifier may come from the internal attachment identifier for the disk drive in the server
4. the machine identifier may come from a server's motherboard or service processor identifier,
5. and the media identifier comes from the volume name used by the server's operating system to identify the volume on the disk drive

Server implementations could control what devices may read and write keys and use the device credential fields to influence access control enforcement.

Another example applied to server virtualization and encryption built into virtualization would be:

1. the virtual machine instance has a unique identifier that is used for the serial number
2. the hypervisor supplies a shared secret that is used as the password to authenticate the virtual machine
3. the network identifier could be used to identify the MAC address of the physical server where the virtual machine is running
4. the machine identifier could be used to identify the hypervisor
5. the media identifier could be used to identify the storage volume used by the virtual machine

These are examples of usage and are not meant to define all device credential usage patterns nor restrict server specific implementations.

The device credentials may be explicitly added by the administrator or may be captured in line with the request and implicitly registered depending upon server policy.

When a server is not able to resolve the identifier values in the device credential to a unique client identification, it may choose to reject the request with an error code of operation failed and reason code of item not found.

## 3.2 Authorization for Revoke, Recover, Destroy and Archive Operations

The authentication suite, as specified in **[KMIP-Prof]**, describes how the client identity is established for KMIP-compliant implementations. This authentication is performed for all KMIP operations.

Certain operations that may be requested by a client via KMIP, particularly Revoke, Recover, Destroy and Archive, may have a significant impact on the availability of a key, on server performance and/or on key security. When a server receives a request for one of these operations, it should ensure that the client has authenticated its identity (see the Authentication Suites section in **[KMIP-Prof]**. The server should also ensure that the client requesting the operation is an object owner, security officer or other identity authorized to issue the request. It may also require additional authentication to ensure that the object owner or a security officer has issued that request. Even with such authentication and authorization, requests for these operations should be considered only a "hint" to the key management system, which may or may not choose to act upon this request depending on server policy.

## 3.3 Using Notify and Put Operations

The Notify and Put operations are the only operations in the KMIP protocol that are initiated by the server, rather than the client. As client-initiated requests are able to perform these functions (e.g., by polling to request notification), these operations are optional for conforming KMIP implementations. However, they provide a mechanism for optimized communication between KMIP servers and clients.

In using Notify and Put, the following constraints and guidelines should be observed:

- The client enrolls with the server, so that the server knows how to locate the client to which a Notify or Put is being sent and which events for the Notify are supported. However, such registration is outside the scope of the KMIP protocol. Registration also includes a specification of whether a given client supports Put and Notify, and what attributes may be included in a Put for a particular client.
- Communication between the client and the server is authenticated. Authentication for a particular client/server implementation is at a minimum accomplished using one of the mandatory authentication mechanisms (see **[KMIP-Prof]**). Further strengthening of the client/server communications integrity by means of signed message content and/or wrapped keys is recommended.
- In order to minimize possible divergence of key or state information between client and server as a result of server-initiated communication, any client receiving Notify or Put messages returns acknowledgements of these messages to the server. This acknowledgement may be at communication layers below the KMIP layer, such as by using transport-level acknowledgement provided in TCP/IP.
- For client devices that are incapable of responding to messages from the server, communication with the server happens via a proxy entity that communicates with the server, using KMIP, on behalf of the client. It is possible to secure communication between a proxy entity and the client using other, potentially proprietary mechanisms.

## 3.4 Usage Allocation

Usage should be allocated and handled carefully at the client, since power outages or other types of client failures (crashes) may render allocated usage lost. For example, in the case of a key being used for the encryption of tapes, such a loss of the usage allocation information following a client failure during encryption may result in the necessity for the entire tape backup session to be re-encrypted using a different key, if the server is not able to allocate more usage. It is possible to address this through such approaches as caching usage allocation information on stable storage at the client, and/or having conservative allocation policies at the server (e.g., by keeping the maximum possible usage allocation per client request moderate). In general, usage allocations should be as small as possible; it is preferable to use multiple smaller allocation requests rather than a single larger request to minimize the likelihood of unused allocation.

  

## 3.5 Key State and Times

**[KMIP-Spec]** provides a number of time-related attributes, including the following:

- Initial Date: The date and time when the managed cryptographic object was first created by or registered at the server.
- Activation Date: The date and time when the managed cryptographic object should begin to be used for applying cryptographic protection to data.
- Process Start Date: The date and time when a managed symmetric key object should begin to be used for processing cryptographically protected data. The managed symmetric key object should not be used prior to this date.
- Protect Stop Date: The date and time when a managed symmetric key object should no longer be used for applying cryptographic protection to data
- Deactivation Date: The date and time when the managed cryptographic object should no longer be used for applying cryptographic protection (e.g., encryption, signing, wrapping, MACing, deriving). Under extraordinary circumstances and when special permission is granted the managed symmetric key object can be used for decryption, signature verification, unwrapping, or MAC verification,
- Destroy Date: The date and time when the managed cryptographic object was destroyed
- Compromise Occurrence Date: The date and time when the managed cryptographic object was first believed to be compromised.
- Compromise Date: The date and time when the managed cryptographic object was entered into the compromised state.
- Archive Date: The date and time when the managed object was placed in Off-Line storage.

These attributes apply to all cryptographic objects (symmetric keys, asymmetric keys, etc.) with exceptions as noted in **[KMIP-Spec]**. However, certain of these attributes (such as the Initial Date) are not specified by the client and are implicitly set by the server.

In using these attributes, the following guidelines should be observed:

- As discussed for each of these attributes in **[KMIP-Spec],** a number of these times are set once and it is not possible for the client or server to modify them. However, several of the time attributes (particularly the Activation Date, Protect Start Date, Process Stop Date and Deactivation Date) may be set by the server and/or requested by the client. Coordination of time-related attributes between client and server, therefore, is primarily the responsibility of the server, as it manages the cryptographic object and its state. However, special conditions related to time-related attributes, governing when the server accepts client modifications to time-related attributes, may be

  

communicated out-of-band between the client and server outside the scope of KMIP. In general, state transitions occur as a result of operational requests, such as Create, Create Key Pair, Register, Activate, Revoke, and Destroy. However, clients may need to specify times in the future for such things as Activation Date, Deactivation Date, Process Start Date, and Protect Stop Date.
KMIP allows clients to specify times in the past for such attributes as Activation Date and Deactivation Date. This is intended primarily for clients that were disconnected from the server at the time that the client performed that operation on a given key.

- It is valid to have a projected Deactivation Date when there is no Activation Date. This means, however, that the key is not yet active, even though its projected Deactivation Date has been specified. A valid Deactivation Date is greater than or equal to the Activation Date (if the Activation Date has been set).
- The Protect Stop Date may be equal to, but may not be later than the Deactivation Date. Similarly, the Process Start Date may be equal to, but may not precede, the Activation Date. KMIP implementations should consider specifying both these attributes, particularly for symmetric keys, as a key may be needed for processing protected data (e.g., decryption) long after it is no longer appropriate to use it for applying cryptographic protection to data (e.g., encryption).
- KMIP does not allow an Active object to be destroyed with the Destroy operation. The server returns an error, if the client invokes the Destroy operation on an Active object. To destroy an Active object, clients first call the Revoke operation or explicitly set the Deactivation Date of the object. Once the object is in Deactivated state, clients may destroy the object by calling the Destroy operation. These operations may be performed in a batch. If other time-related attributes (e.g., Protect Stop Date) are set to a future date, the server should set these to the Deactivation Date.
- After a cryptographic object is destroyed, a key management server may retain certain information about the object, such as the Unique Identifier.

KMIP allows the specification of attributes on a per-client basis, such that a server could maintain or present different sets of attributes for different clients. This flexibility may be necessary in some cases, such as when a server maintains the availability of a given key for some clients, even after that same key is moved to an inactive state (e.g., Deactivated state) for other clients. However, such an approach might result in significant inconsistencies regarding the object state from the point of view of all participating clients and should, therefore, be avoided. A server should maintain a consistent state for each object, across all clients that have or are able to request that object.

## 3.6 Template

The usage of templates is an alternative approach for setting attributes in an operation request. Instead of individually specifying each attribute, a template may be used to provide attribute values.

  

A template also has attributes that are applicable to the template itself which are referred to in the specification as *associated attributes* to distinguish them from the attributes that are contained within the template managed object. When registering a template, the Name attribute for the template itself must be set. It is used to identify the template in the Template-Attribute structure when attributes for a managed object are set in KMIP operations.

The Template-Attribute structure allows for multiple template names (zero or more) and individual attributes (zero or more) to be specified in an operation request. The structure is used in the Create, Create Key Pair, Register, Re-key, Re-key Key Pair, Derive Key, Certify, and Re-certify operations. All of these operations with the exception of the Create Key Pair and the Re-key Key Pair operations use the Template-Attribute tag. The Create Key Pair and the Re-key Key Pair operations use the Common Template-Attribute, Private Key Template Attribute, and Public Key Template-Attribute tags allowing specification of different attributes for the public and private managed cryptographic objects.

Templates may be the subject of the Register, Locate, Get, Get Attributes, Get Attribute List, Add Attribute, Modify Attribute, Delete Attribute, Delete Attribute, and Destroy operations. Templates are created using the Register operation. When the template is the subject of an operation, the Unique Identifier is used to identify the template. The template name is only used to identify the template when referenced inside a Template-Attribute structure.

### 3.6.1 Template Usage Examples

The purpose of these examples is to illustrate how templates are used. The first example shows how a template is registered. The second example shows how the newly registered template is used to create a symmetric key.

#### 3.6.1.1.1 Example of Registering a Template

In this example, a client registers a template by encapsulating attributes for creating a 256-bit AES key with the Cryptographic Usage Mask set to Encrypt and Decrypt.

The following is specified inside the Register Request Payload:

- Object Type: Template
- Template-Attribute:
  - Attribute
    - Attribute Name : Name
    - Attribute Value: Template1
- Template
  - Attribute
    - Attribute Name: Cryptographic Algorithm
    - Attribute Value: AES
  - Attribute
    - Attribute Name: Cryptographic Length

  

        - Attribute Value: 256
    - Attribute
        - Attribute Name: Cryptographic Usage Mask
        - Attribute Value: Encrypt and Decrypt
    - Attribute
        - Attribute Name: Operation Policy Name
        - Attribute Value: OperationPolicy1

The Operation Policy OperationPolicy1 applies to the AES key being created using the template. It is not used to control operations on the template itself. KMIP does not allow operation policies to be specified for controlling operations on the template itself. The default policy for template objects is used for this purpose and is specified in the KMIP Specification.

#### 3.6.1.2 Example of Creating a Symmetric Key using a Template

In this example, the client uses the template created in example 3.6.1 to create a 256-bit AES key.

The following is specified in the Create Request Payload:

- Object Type: Symmetric Key
- Template-Attribute:
    - Name: Template1
    - Attribute:
        - Attribute Name: Name
        - Attribute Value: AESkey
    - Attribute:
        - Attribute Name: x-Custom Attribute1
        - Attribute Value: ID74592

The Template-Attribute structure specifies both a template name and additional associated attributes. It is possible to specify the Custom Attribute inside the template when the template is registered; however, this particular example sets this attribute separately.

#### 3.6.1.3 Compatibility Note:

Versions of KMIP prior to KMIP version 1.2 contained a fixed list of attributes applicable to objects created using a template and those applicable to the template managed object. The value returned by the Get operation for a template was subject to varying interpretations. KMIP 1.2 alters this handling to provide clarification of the expected handling for templates. KMIP clients may need to be mindful of this change when registering or performing operations which refer to templates as the handling of templates in a KMIP server vary depending on the version of the KMIP protocol specified.

  

As the baseline server profile does not mandate (require) support for templates a KMIP client that requires support for templates cannot be guaranteed to interoperate with all servers that conform to the KMIP specification.

## 3.7 Archive Operations

When the Archive operation is performed, it is recommended that a unique identifier and a minimal set of attributes be retained within the server for operational efficiency. In such a case, the retained attributes may include Unique Identifier and State.

## 3.8 Message Extensions

Any number of vendor-specific extensions may be included in the Message Extension optional structure. This allows KMIP implementations to create multiple extensions to the protocol.

## 3.9 Unique Identifiers

For clients that require unique identifiers in a special form, out-of-band registration/configuration may be used to communicate this requirement to the server.

## 3.10 Result Message Text

KMIP specifies the Result Status, the Result Reason and the Result Message as normative message contents. For the Result Status and Result Reason, the enumerations provided in **[KMIP-Spec]** are the normative values. The values for the Result Message text are implementation-specific. In consideration of internationalization, it is recommended that any vendor implementation of KMIP provide appropriate language support for the Return Message. How a client specifies the language for Result Messages is outside the scope of the KMIP.

## 3.11 Query

Query does not explicitly support client requests to determine what operations require authentication. To determine whether an operation requires authentication, a client should request that operation.

## 3.12 Canceling Asynchronous Operations

If an asynchronous operation is cancelled by the client, no information is returned by the server in the result code regarding any operations that may have been partially completed. Identification and remediation of partially completed operations is the responsibility of the server.

It is the responsibility of the server to determine when to discard the status of asynchronous operations. The determination of how long a server should retain the status of an asynchronous operation is implementation-dependent and not defined by KMIP.

  

Once a client has received the status on an asynchronous operation other than “pending”, any subsequent request for status of that operation may return either the same status as in a previous polling request or an “unavailable” response.

## 3.13 Multi-instance Hash

The Digest attribute contains the output of hashing a managed object, such as a key or a certificate. The server always generates the SHA-256 hash value when the object is created or generated. KMIP allows multiple instances of the digest attribute to be associated with the same managed object. For example, it is common practice for publicly trusted CAs to publish two digests (often referred to as the fingerprint or the thumbprint) of their certificate: one calculated using the SHA-1 algorithm and another using the MD5 algorithm. In this case, each digest would be calculated by the server using a different hash algorithm.

## 3.14 Returning Related Objects

The key block returns a single object, with associated attributes and other data. For those cases in which multiple related objects are needed by a client, such as the private key and the related certificate, the client should issue multiple Get requests to obtain these related objects.

## 3.15 Reducing Multiple Requests through the Use of Batch

KMIP supports batch operations in order to reduce the number of calls between the client and server. For example, Locate and Get are likely to be commonly accomplished within a single batch request.

KMIP does not ensure that batch operations are atomic on the server side. If servers implement such atomicity, the client is able to use the optional “undo” mode to request roll-back for batch operations implemented as atomic transactions. However, support for “undo” mode is optional in the protocol, and there is no guarantee that a server that supports “undo” mode has effectively implemented atomic batches. The use of “undo”, therefore, should be restricted to those cases in which it is possible to assure the client, through mechanisms outside of KMIP, of the server effectively supporting atomicity for batch operations.

## 3.16 Maximum Message Size

When a server is processing requests in a batch, it should compare the cumulative response size of the message to be returned after each request with the specified Maximum Response Size. If the message is too large, it should prepare a maximum message size error response message at that point, rather than continuing with operations in the batch. This increases the client’s ability to understand what operations have and have not been completed.

When processing individual requests within the batch, the server that has encountered a Maximum Response Size error should not return attribute values or other information as part of the error response.

The Locate operation also supports the concept of a maximum item count to include in the returned list of unique identifiers.

  

## 3.17 Using Offset in Re-key and Re-certify Operations

The Re-key, Re-key Key Pair, and Re-certify operations allow the specification of an offset interval.

The Re-key and the Re-key Key Pair operations allow the client to specify an offset interval for activation of the key. This offset specifies the duration of time between the time the request is made and the time when the activation of the key occurs. If an offset is specified, all other times for the new key are determined from the new Activation Date, based on the intervals used by the previous key, i.e., from the Activation Date to the Process Start Date, Protect Stop Date, etc.

The Re-certify operation allows the client to specify an offset interval that indicates the difference between the Initial Date of the new certificate and the Activation Date of the new certificate. As with the Re-key operation, all other times for the certificate are determined using the intervals used for the previous certificate.

Note that in re-key operations if activation date, process start date, protect stop date and deactivation date are obtained from the existing key, and the initial date is obtained from the current time, then the deactivation/activation date/process start date/protect stop date is smaller or less than initial date. KMIP allows back-dating of these values to prevent this contradiction (see **[KMIP-Spec]** section 3.22).

## 3.18 ID Placeholder

A number of operations are affected by a mechanism referred to as the ID Placeholder. This is a temporary variable consisting of a single Unique Identifier that is stored inside the server for the duration of executing a batch of operations. The ID Placeholder is obtained from the Unique Identifier returned by certain operations; the applicable operations are identified in Table 1, along with a list of operations that accept the ID Placeholder as input.

| **Operation** | **ID Placeholder at the beginning of the operation** | **ID Placeholder upon completion of the operation (in case of operation failure, a batch using the ID Placeholder stops)** |
|---|---|---|
| Create | - | ID of new Object |
| Create Key Pair | - | ID of new Private Key (ID of new Public Key may be obtained via a Locate) |
| Create Split Key | - | ID of the split whose Key Part Identifier is 1 |
| Join Split Key | | ID of returned object |
| Register | - | ID of newly registered Object |
| Derive Key | - (multiple Unique Identifiers may be specified in the | ID of new Symmetric Key |

  

| | | |
|---|---|---|
| | request) | |
| Locate | - | ID of located Object |
| Get | ID of Object | no change |
| Validate | - | - |
| Get Attributes List/Modify/Add/Delete | ID of Object | no change |
| Activate | ID of Object | no change |
| Revoke | ID of Object | no change |
| Destroy | ID of Object | no change |
| Archive/Recover | ID of Object | no change |
| Certify | ID of Public Key | ID of new Certificate |
| Re-certify | ID of Certificate | ID of new Certificate |
| Re-key | ID of Symmetric Key to be rekeyed | ID of new Symmetric Key |
| Re-key Key Pair | ID of Private Key to be rekeyed | ID of new Private Key (ID of new Public Key may be obtained via a Locate) |
| Obtain Lease | ID of Object | no change |
| Get Usage Allocation | ID of Key | no change |
| Check | ID of Object | no change |

**Table 1: ID Placeholder Prior to and Resulting from a KMIP Operation**

## 3.19 Key Block

The protocol uses the Key Block structure to transport a key to the client or server. This Key Block consists of the Key Value Type, the Key Value, and the Key Wrapping Data. The Key Value Type identifies the format of the Key Material, e.g., Raw format or Transparent Key structure. The Key Value consists of the Key Material and optional attributes. The Key Wrapping Data provides information about the wrapping key and the wrapping mechanism, and is returned only if the client requests the Key Value to be wrapped by specifying the Key Wrapping Specification inside the Get Request Payload. The Key Wrapping Data may also be included inside the Key Block if the client registers a wrapped key.

The protocol allows any attribute to be included inside the Key Value and allows these attributes to be cryptographically bound to the Key Material (i.e., by signing, MACing, encrypting, or both encrypting and signing/MACing the Key Value). Some of the attributes that may be included include the following:

- Unique Identifier – uniquely identifies the key
- Cryptographic Algorithm (e.g., AES, 3DES, RSA) – this attribute is either specified inside the Key Block structure or the Key Value structure
- Cryptographic Length (e.g., 128, 256, 2048) – this attribute is either specified inside the Key Block structure or the Key Value structure
- Cryptographic Usage Mask– identifies the cryptographic usage of the key (e.g., Encrypt, Wrap Key, Export)
- Cryptographic Parameters – provides additional parameters for determining how the key may be used
    - Block Cipher Mode (e.g., CBC, NISTKeyWrap, GCM) – this parameter identifies the mode of operation, including block cipher-based MACs or wrapping mechanisms
    - Padding Method (e.g., OAEP, X9.31, PSS) – identifies the padding method and if applicable the signature or encryption scheme
    - Hashing Algorithm (e.g., SHA-256) – identifies the hash algorithm to be used with the signature/encryption mechanism or Mask Generation Function; note that the different HMACs are defined individually as algorithms and do not require the Hashing Algorithm parameter to be set
    - Key Role Type – Identifies the functional key role (e.g., DEK, KEK)
- State (e.g., Active)
- Dates (e.g., Activation Date, Process Start Date, Protect Stop Date)
- Custom Attribute – allows vendors and clients to define vendor-specific attributes; may also be used to prevent replay attacks by setting a nonce

## 3.20 Object Group

The key management system may specify rules for valid group names which may be created by the client. Clients are informed of such rules by a mechanism that is not specified by **[KMIP-Spec]**KMIP_Spec. In the protocol, the group names themselves are text strings of no specified format. Specific key management system implementations may choose to support hierarchical naming schemes or other syntax restrictions on the names. Groups may be used to associate objects for a variety of purposes. A set of keys used for a common purpose, but for different time intervals, may be linked by a common Object Group. Servers may create predefined groups and add objects to them independently of client requests.

KMIP allows clients to specify whether it wants a "fresh" or "default" object from a common Object Group. Fresh is an indication of whether a member of a group has been retrieved by a client with the Get operation. The value of fresh may be set as an attribute when creating or registering an object. Subsequently, the Fresh attribute is modifiable only by the server. For example, a set of symmetric keys belong to the Object Group "SymmetricKeyGroup1" and the Fresh attribute is set to true for members of the group at the time of creating or registering the member. To add a new symmetric key to the group, the Object Group attribute is set to

  

“SymmetricKeyGroup1” and the Fresh attribute is set to true when creating or registering the symmetric key object.

The definition of a “default” object in a group is based on server policy. One example of server policy is to use round robin selection to serve a key from a group. In this case when a client requests the default key from a group, the server uses round robin selection to serve the key.

An object may be removed from a group by deleting the Object Group attribute, as long as server policy permits it. A client would need to delete each individual member of a group to remove all members of a group.

The Object Group Member flag is specified in the Locate request to indicate the type of group member to return. Object Group Member is an enumeration that can take the value Group Member Fresh or Group Member Default. Following are examples of how the Object Group Member flag is used:

When a Locate request is made by specifying the Object Group attribute (e.g., "symmetricKeyGroup1) and setting the Object Group Member flag to "Group Member Fresh", matching objects from the specified group (e.g., "symmetricKeyGroup1") have the Fresh attribute set to true. If there are no fresh objects remaining in the group, the server may generate a new object on the fly based on server policy.

When a Locate request is made by specifying the Object Group attribute (e.g., "symmetricKeyGroup2) and setting the Object Group Member flag to "Group Member Default", a default object is returned from the group. In this example, the server policy defines default to be the next key in the group "symmetricKeyGroup2”; the group has three group members whose Unique Identifiers are uuid1, uuid2, uuid3. If the client performs four consecutive batched Locate and Get operations with Object Group set to "symmetricKeyGroup2" and Object Group Member set to “Group Member Default” in the Locate request, the server returns uuid1, uuid2, uuid3, and uuid1 (restarting from the beginning with uuid1 for the fourth request) in the four Get responses.

## 3.21 Certify and Re-certify

The key management system may contain multiple embedded CAs or may have access to multiple external CAs. How the server routes a certificate request to a CA is vendor-specific and outside the scope of KMIP. If the server requires and supports the capability for clients to specify the CA to be used for signing a Certificate Request, then this information may be provided by including the X.509 Certificate Issuer attribute in the Certify or Re-certify request.

**[KMIP-Spec]** supports multiple options for submitting a certificate request to the key management server within a Certify or Re-Certify operation. It is a vendor decision as to whether the key management server offers certification authority (CA) functionality or proxies the certificate request onto a separate CA for processing. The type of certificate request formats supported is also a vendor decision, and this may, in part, be based upon the request formats supported by any CA to which the server proxies the certificate requests.

  

All certificate request formats for requesting X.509 certificates specified in **[KMIP-Spec]** (i.e., PKCS#10, PEM and CRMF) provide a means for allowing the CA to verify that the client that created the certificate request possesses the private key corresponding to the public key in the certificate request. This is referred to as Proof-of-Possession (POP). However, it should be noted that in the case of the CRMF format, some CAs may not support the CRMF POP option, but instead rely upon the underlying certificate management protocols (i.e., CMP and CMC) to provide POP.  In the case where the CA does not support POP via the CRMF format (including CA functionality within the key management server), an alternative certificate request format (i.e., PKCS#10, PEM) would need to be used if POP needs to be verified.

## 3.22 Specifying Attributes during a Create Key Pair or Re-key Key Pair Operation

The Create Key Pair and the Re-key Key Pair operations allow clients to specify attributes using the Common Template-Attribute, Private Key Template-Attribute, and Public Key Template-Attribute. The Common Template-Attribute object includes a list of attributes that apply to both the public and private key. Attributes that are not common to both keys may be specified using the Private Key Template-Attribute or Public Key Template-Attribute. If a single-instance attribute is specified in multiple Template-Attribute objects, the server obeys the following order of precedence:

1. Attributes specified explicitly in the Private and Public Key Template-Attribute, then
2. Attributes specified via templates in the Private and Public Key Template-Attribute, then
3. Attributes specified explicitly in the Common Template-Attribute, then
4. Attributes specified via templates in the Common Template-Attribute

### 3.22.1 Example of Specifying Attributes during the Create Key Pair Operation

A client specifies several attributes in the Create Key Pair Request Payload. The Common Template-Attribute includes the template name RSACom and other explicitly specified common attributes:

<u>RSACom Template</u>

- Template
    - Attribute
        - Attribute Name: Cryptographic Algorithm
        - Attribute Value: RSA
    - Attribute
        - Attribute Name: Cryptographic Length
        - Attribute Value: 2048
    - Attribute
        - Attribute Name: Cryptographic Parameters
        - Attribute Value:

  

        - Padding Method: OAEP
- Attribute:
    - Attribute Name: x-Serial
    - Attribute Value: 1234
- Attribute:
    - Attribute Name: Object Group:
    - Attribute Value: Key encryption group 1

<u>Common Template-Attribute</u>

- Name
    - Name Value: RSACom
    - Name Type: Uninterpreted Text String
- Attribute
    - Attribute Name: Cryptographic Length:
    - Attribute Value: 4096
- Attribute
    - Attribute Name: Cryptographic Parameters
    - Attribute Value:
        - Padding Method: PKCS1 v1.5
- Attribute
    - Attribute Name: x-ID
    - Attribute Value: 56789

The Private Key Template-Attribute includes a reference to the template name RSAPriv and other explicitly-specified private key attributes:

<u>RSAPriv Template</u>

- Template
    - Attribute
        - Attribute Name: Object Group
        - Attribute Value: Key encryption group 2

<u>Private Key Template-Attribute</u>

- Name
    - Name Value: RSAPriv
    - Name Type: Uninterpreted Text String
- Attribute
    - Attribute Name: Cryptographic Usage Mask

  

    - Attribute Value: Unwrap Key
- Attribute
    - Attribute Name: Name
    - Attribute Value:
        - Name Value: PrivateKey1
        - Name Type:  Uninterpreted Text String

The Public Key Template Attribute includes explicitly-specified public key attributes:

<u>Public Key Template-Attribute</u>

- Attribute
    - Attribute Name: Cryptographic Usage Mask
    - Attribute Value: Wrap Key
- Attribute
    - Attribute Name: Name
    - Attribute Value:
        - Name Value: PublicKey1
        - Name Type:  Uninterpreted Text String

Following the attribute precedence rule, the server creates a 4096-bit RSA key. The following client-specified attributes are set:

<u>Private Key</u>

- Cryptographic Algorithm: RSA
- Cryptographic Length: 4096
- Cryptographic Parameters:
    - Padding Method: OAEP
- Cryptographic Parameters:
    - Padding Method: PKCS1 v1.5
- Cryptographic Usage Mask: Unwrap Key
- x-Serial: 1234
- x-ID: 56789
- Object Group: Key encryption group 1
- Object Group: Key encryption group 2
- Name:
    - Name Value: PrivateKey1
    - Name Type: Uninterpreted Text String

<u>Public Key</u>

- Cryptographic Algorithm: RSA

  

- Cryptographic Length: 4096
- Cryptographic Parameters:
    - Padding Method: OAEP
- Cryptographic Parameters:
    - Padding Method: PKCS1 v1.5
- Cryptographic Usage Mask: Wrap Key
- x-Serial: 1234
- x-ID: 56789
- Object Group: Key encryption group 1
- Name:
    - Name Value: PublicKey1
    - Name Type: Uninterpreted Text String

## 3.23 Registering a Key Pair

During a Create Key Pair or Re-key Key Pair operation, a Link Attribute is automatically created by the server for each object (i.e., a link is created from the private key to the public key and vice versa). Certain attributes are the same for both objects and are set by the server while creating the key pair. The KMIP protocol does not support an equivalent operation for registering a key pair. Clients are able to register the objects independently and manually set the Link attributes to make the server aware that these keys are associated with each other. When the Link attribute is set for both objects, the server should verify that the registered objects indeed correspond to each other and apply similar restrictions as if the key pair was created on the server.

Clients should perform the following steps when registering a key pair:

1. Register the public key and set all associated attributes:
    a. Cryptographic Algorithm
    b. Cryptographic Length
    c. Cryptographic Usage Mask
5. Register the private key and set all associated attributes
    a. Cryptographic Algorithm is the same for both public and private key
    b. Cryptographic Length is the same for both public and private key
    c. Cryptographic Parameters may be set; if set, the value is the same for both the public and private key
    d. Cryptographic Usage Mask is set, but does not contain the same value for both the public and private key
    e. Link is set for the Private Key with Link Type *Public Key Link* and the Linked Object Identifier of the corresponding Public Key
    f. Link is set for the Public Key with Link Type *Private Key Link* and the Linked Object Identifier of the corresponding Private Key

  

## 3.24 Non-Cryptographic Objects

The KMIP protocol allows clients to register Secret Data objects. Secret Data objects may include passwords or data that are used to derive keys.

KMIP defines Secret Data as cryptographic objects. Even if the object is not used for cryptographic purposes, clients may still set certain attributes, such as the Cryptographic Usage Mask, for this object unless otherwise stated. Similarly, servers set certain attributes for this object, including the Digest, State, and certain Date attributes, even if the attributes may seem relevant only for other types of cryptographic objects.

When registering a Secret Data object, the following attributes are set by the server:

- Unique Identifier
- Object Type
- Digest
- State
- Initial Date
- Last Change Date

When registering a Secret Data object for non-cryptographic purposes, the following attributes are set by either the client or the server:

- Cryptographic Usage Mask

## 3.25 Asymmetric Concepts with Symmetric Keys

The Cryptographic Usage Mask attribute is intended to support asymmetric concepts using symmetric keys. This is common practice in established crypto systems: the MAC is an example of an operation where a single symmetric key is used at both ends, but policy dictates that one end may only generate cryptographic tokens using this key (the MAC) and the other end may only verify tokens. The security of the system fails if the verifying end is able to use the key to perform generation operations.

In these cases it is not sufficient to describe the usage policy on the keys in terms of cryptographic primitives like “encrypt” vs. “decrypt” or “sign” vs. “verify”. There are two reasons why this is the case.

- In some of these operations, such as MAC generation and verification, the same cryptographic primitive is used in both of the complementary operations. MAC generation involves computing and returning the MAC, while MAC verification involves computing that same MAC and comparing it to a supplied value to determine if they are the same. Thus, both generation and verification use the “encrypt” operation, and the two usages are not able to be distinguished by considering only “encrypt” vs. “decrypt”.
- Some operations which require separate key types use the same fundamental cryptographic primitives. For example, encryption of data, encryption of a key, and computation of a MAC all use the fundamental operation “encrypt”, but in many applications, securely differentiated keys are used for these three operations. Simply looking for an attribute that permits “encrypt” is not sufficient.

  

Allowing the use of these keys outside of their specialized purposes may compromise security. Instead, specialized application-level permissions are necessary to control the use of these keys. KMIP provides several pairs of such permissions in the Cryptographic Usage Mask (3.14), such as:

| | |
|---|---|
| MAC GENERATE<br>MAC VERIFY | For cryptographic MAC operations. Although it is possible to compose certain MACs using a series of encrypt calls, the security of the MAC relies on the operation being atomic and specific. |
| GENERATE CRYPTOGRAM<br>VALIDATE CRYPTOGRAM | For composite cryptogram operations such as financial CVC or ARQC. To specify exactly which cryptogram the key is used for it is also necessary to specify a *role* for the key (see Section 3.6 “Cryptographic Parameters” in **[KMIP-Spec]**). |
| TRANSLATE ENCRYPT<br>TRANSLATE DECRYPT<br>TRANSLATE WRAP<br>TRANSLATE UNWRAP | To accommodate secure routing of traffic and data. In many areas that rely on symmetric techniques (notably, but not exclusively financial networks), information is sent from place to place encrypted using shared symmetric keys. When encryption keys are changed, it is desirable for the change to be an atomic operation, otherwise distinct unwrap-wrap or decrypt-encrypt steps risk leaking the plaintext data during the translation process.<br>*TRANSLATE ENCRYPT/DECRYPT* is used for data encipherment.<br>*TRANSLATE WRAP/UNWRAP* is used for key wrapping. |

**Table 2: Cryptographic Usage Masks Pairs**

In order to support asymmetric concepts using symmetric keys in a KMIP system, the server implementation needs to be able to differentiate between clients for generate operations and clients for verify operations. As indicated by Section 3 (“Attributes”) of **[KMIP-Spec]** there is a single key object in the system to which all relevant clients refer, but when a client requests that key, the server is able to choose which attributes (permissions) to send with it, based on the identity and configured access rights of that specific client. There is, thus, no need to maintain and synchronize distinct copies of the symmetric key – just a need to define access policy for each client or group of clients.

The internal implementation of this feature at the server end is a matter of choice for the vendor: storing multiple key blocks with all necessary combinations of attributes or generating key blocks dynamically are both acceptable approaches.

  

## 3.26 Application Specific Information

The Application Specific Information attribute is used to store data which is specific to the application(s) using the object. Some examples of Application Namespace and Application Data pairs are given below.

- SMIME, 'someuser@company.com'
- TLS, 'some.domain.name'
- Volume Identification, '123343434'
- File Name, 'secret.doc'
- Client Generated Key ID, ‘450994003'

The following Application Namespaces are recommended:

- SMIME
- TLS
- IPSEC
- HTTPS
- PGP
- Volume Identification
- File Name
- LTO4, LTO5, and LTO6
- LIBRARY-LTO, LIBRARY-LTO4, LIBRARY-LTO5 and LIBRARY-LTO6

KMIP provides optional support for server-generated Application Data. Clients may request the server to generate the Application Data for the client by omitting Application Data while setting or modifying the Application Specific Information attribute. A server only generates the Application Data if the Application Data is completely omitted from the request, and the client-specified Application Namespace is recognized and supported by the server. An example for requesting the server to generate the Application Data is shown below:

AddAttribute(Unique ID, AppSpecInfo{AppNameSpace=’LIBRARY-LTO4’});

If the server does not recognize the namespace, the “Application Namespace Not Supported” error is returned to the client.

If the Application Data is provided, and the Application Namespace is recognized by the server, the server uses the provided  Application Data, and does not generate the Application Data for the client. In the example below, the server stores the Application Specific Information attribute with the Application Data value set to null.

AddAttribute(Unique ID, AppSpecInfo{AppNameSpace=’LIBRARY-LTO4’, AppData=null});

## 3.27 Mutating Attributes

KMIP does not support server mutation of client-supplied attributes. If a server does not accept an attribute value that is being specified inside the request by the client, the server returns an error and specifies “Invalid Field” as Result Reason.

  

Attributes that are not set by the client, but are implicitly set by the server as a result of the operation, may optionally be returned by the server in the operation response inside the Template–Attribute.

If a client sets a time-related attribute to the current date and time (as perceived by the client), but as a result of a clock skew, the specified date of the attribute is earlier than the time perceived by the server, the server's policy is used to determine whether to accept the "backdated attribute". KMIP does not require the server to fail a request if a backdated attribute is set by the client.

If a server does not support backdated attributes, and cryptographic objects are expected to change state at the specified current date and time (as perceived by the client), clients are recommended to issue the operation that would implicitly set the date for the client. For example, instead of explicitly setting the Activation Date, clients could issue the Activate operation. This would require the server to set the Activation Date to the current date and time as perceived by the server.

If it is not possible to set a date attribute via an operation, and the server does not support backdated attributes, clients need to take into account that potential clock skew issues may cause the server to return an error even if a date attribute is set to the client's current date and time.

For additional information, refer to the sections describing the State attribute and the Time Stamp field in **[KMIP-Spec]**.

## 3.28 Revocation Reason Codes

The enumerations for the Revocation Reason attribute specified in KMIP (see table 9.1.3.2.19 in **[KMIP-Spec]**) are aligned with the Reason Code specified in **[X.509]** and referenced in **[RFC5280]** with the following exceptions. The *certificateHold* and *removeFromCRL* reason codes have been excluded from **[KMIP-Spec]** since KMIP does not support certificate suspension (putting a certificate hold) or unsuspension (removing a certificate from hold). The *aaCompromise* reason code has been excluded from **[KMIP-Spec]** since it only applies to attribute certificates, which are out-of-scope for **[KMIP-Spec]**. The *privilegeWithdrawn* reason code is included in **[KMIP-Spec]** since it may be used for either attribute or public key certificates. In the context of its use within KMIP it is assumed to only apply to public key certificates.

## 3.29 Certificate Renewal, Update, and Re-key

The process of generating a new certificate to replace an existing certificate may be referred to by multiple terms, based upon what data within the certificate is changed when the new certificate is created. In all situations, the new certificate includes a new serial number and new validity dates **[KMIP-Spec]** uses the following terminology which is aligned with the definitions found in IETF **[RFC3647]** and **[RFC4949]:**

- *Certificate Renewal*: The issuance of a new certificate to the subject without changing the subject public key or other information (except the serial number and certificate validity dates) in the certificate.

  

- *Certificate Update:* The issuance of a new certificate, due to changes in the information in the certificate other than the subject public key.
- *Certificate Rekey*: The generation of a new key pair for the subject and the issuance of a new certificate that certifies the new public key.

The KMIP Specification supports certificate renewals using the Re-Certify operation and certificate updates using the Certify operation. Certificate rekey is supported through the submission of a Re-key Key Pair operation, which generates a replacement (new) key pair, followed by a Certify operation, which issues a new certificate containing the replacement (new) public key.

## 3.30 Key Encoding

Two parties receiving the same key as a Key Value Byte String make use of the key in exactly the same way in order to interoperate. To ensure that, it is necessary to define a correspondence between the abstract syntax of Key and the notation in the standard algorithm description that defines how the key is used. The next sections establish that correspondence for the algorithms AES **[FIPS 197]** and Triple-DES **[SP800-67]**.

AES Key Encoding **[FIPS 197]** section 5.2, titled Key Expansion, uses the input key as an array of bytes indexed starting at 0. The first byte of the Key becomes the key byte in AES that is labeled index 0 in **[FIPS 197]** and the other key bytes follow in index order.

Proper parsing and key load of the contents of the Key for AES is determined by using the following Key byte string to generate and match the key expansion test vectors in **[FIPS 197]** Appendix A for the 128-bit (16 byte) AES Cipher Key: 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C.

### 3.30.1 Triple-DES Key Encoding

A Triple-DES key consists of three keys for the cryptographic engine (Key1, Key2, and Key3) that are each 64 bits (even though only 56 are used); the three keys are also referred to as a key bundle (KEY) **[SP800-67]**. A key bundle may employ either two or three mutually independent keys. When only two are employed (called two-key Triple-DES), then Key1 = Key3.

Each key in a Triple-DES key bundle is expanded into a key schedule according to a procedure defined in **[SP800-67]** Appendix A. That procedure numbers the bits in the key from 1 to 64, with number 1 being the left most, or most significant bit. The first byte of the Key is bits 1 through 8 of Key1, with bit 1 being the most significant bit. The second byte of the Key is bits 9 through 16 of Key1, and so forth, so that the last byte of the KEY is bits 57 through 64 of Key3 (or Key2 for two-key Triple-DES).

Proper parsing and key load of the contents of Key for Triple-DES is determined by using the following Key byte string to generate and match the key expansion test vectors in **[SP800-67]** Appendix B for the key bundle:

Key1 = 0123456789ABCDEF

Key2 = 23456789ABCDEF01

  

Key3 = 456789ABCDEF0123

## 3.31 Using the Same Asymmetric Key Pair in Multiple Algorithms

There are mathematical relationships between certain asymmetric cryptographic algorithms such as the Digital Signature Algorithm (DSA) and Diffie-Hellman (DH) and their elliptic curve equivalents ECDSA and ECDH that allow the same asymmetric key pair to be used in both algorithms. In addition, there are overlaps in the key format used to represent the asymmetric key pair for each algorithm type.

Even though a single key pair may be used in multiple algorithms, the KMIP Specification has chosen to specify separate key formats for representing the asymmetric key pair for use in each algorithm. This approach keeps KMIP in line with the reference standards (e.g., NIST **[FIPS 186-4]**, ANSI **[X9.42]**, etc.) from which the key formats are obtained and the best practice documents (e.g., NIST **[SP800-57-1]**, NIST **[SP800-56A]** etc.) which recommend that a key pair only be used for one purpose.

## 3.32 Cryptographic Length of Asymmetric Keys

The value (e.g., 2048 bits) referred to in the KMIP *Cryptographic Length* attribute for an asymmetric (public or private) key may be misleading, since this length only refers to certain portions of the mathematical values that comprise the key. The actual length of all the mathematical values comprising the public or the private key is longer than the referenced value. This point may be illustrated by looking at the components of a RSA public and private key.

The RSA public key is comprised of a modulus (n) and an (public) exponent (e). When one indicates that the RSA public key is 2048 bits in length that is a reference to the bit length of the modulus (n) only. So the full length of the RSA public key is actually longer than 2048 bits, since it also includes the length of the exponent (e) and the overhead of the encoding (e.g., ASN.1) of the key material.

The RSA private key is comprised of a modulus (n), the public exponent (e), the private exponent (d), prime 1 (p), prime 2 (q), exponent 1 (d mod (p-1)), exponent 2 (d mod (p-1)), and coefficient ((inverse of q) mod p). Once again the 2048 bit key length is referring only to the length of the modulus (n), so the overall length of the private key would be longer given the number of additional components which comprise the key and the overhead of encoding (e.g., ASN.1) of the key material.

KMIP implementations need to ensure they do not make assumptions about the actual length of asymmetric (public and private) key material based on the value specified in the *Cryptographic Length* attribute.

## 3.33 Discover Versions

The Discover Versions operation allows clients and servers to identify a KMIP protocol version that both client and server understand. The operation was added to KMIP 1.1. KMIP 1.0 clients and servers may therefore not support this operation. If the Discover Versions request is sent to a KMIP 1.0 server and the server does not support the operation, the server returns the "Operation Not Supported" error.

The operation addresses both the "dumb" and "smart" client scenarios. Dumb clients may simply pick the first protocol version that is returned by the server, assuming that the client provides the server with a list of supported protocol version. Smart clients may request the server to return a complete list of supported protocol versions by sending an empty request payload and picking a protocol version that is supported by both client and server.

Clients specify the protocol version in the request header and optionally provide a list of protocol versions in the request payload. If the protocol version in the request header is not specified in the request payload and the server does not support any protocol version specified in the request payload, the server returns an empty list in the response payload. In this scenario, clients are aware that the request did not result in an error and could communicate with the server using the protocol version specified in the request header.

## 3.34 Vendor Extensions

KMIP allows for vendor extensions in a number of areas:

1. Enumerations have specific ranges which are noted as extensions
2. Item Tag values of the form 0x54xxxx are reserved for vendor extensions
3. Attributes may be defined by the client with a "x-" prefix or by the server with a "y-" prefix

Extensions may be used by vendors to communicate information between a KMIP client and a KMIP server that is not currently defined within the KMIP specification.

A common use of extensions is to allow for the structured definition of attributes using KMIP TTLV encoding rather than encoding vendor specific information in opaque byte strings.

## 3.35 Certificate Revocation Lists

Any Certificate Revocation List (CRL) checking which may be required for certificate-related operations such as register and re-key should be performed by the client prior to requesting the operation from a server.

## 3.36 Using the "Raw" Key Format Type

As defined in Section 2.1.3 of the KMIP Specification V1.1, the "raw" key format is intended to be used for "a key that contains only cryptographic key material, encoded as a string of bytes. The "raw" key format supports situations such as "non-KMIP-aware end-clients are aware how wrapped cryptographic objects (possibly Raw keys) from the KMIP server should be used without having to rely on the attributes provided by the Get Attributes operation" and in that regard is similar to the Opaque key format type. "Raw" key format is intended to be applied to symmetric keys and not asymmetric keys; therefore, this format is not specified in the asymmetric key profiles included in KMIP V1.1.

  

## 3.37 Use of Meta-Data Only (MDO) Keys

Meta-Data Only ( MDO) keys are those Managed Key Objects for which no Key Value is present, as introduced in version 1.2 of **[KMIP-Spec]** MDO objects can be one of the following: Symmetric Keys, Private Keys, Split Keys, or Secret Data.

This may be a result of the KMIP client only wanting to register information (Meta-Data) about the key with a Key Management System, without having the key itself leave the client's physical boundary. One such example could be for keys created and stored within a Hardware Security Module (HSM), with a policy that does not allow for the keys to leave its hardware. In such cases, the KMIP client will not include a Key Value within the Key Block during a Register operation, although it may optionally include a Key Value Location attribute indicating the location of the Key Value instead. For such keys, as part of the Register operation, the server will create a Key Value Present attribute and set it to false to indicate the key value is not stored on the server.

The KMIP protocol does not support the addition of a Key Value to an existing MDO key object on the server. If for some reason the client wanted to do this, it would have to carry out another Register operation and create a new managed object with the Key Value.

Finally, because there is no Key Value associated with an MDO key on the server, KMIP operations for Re-key, Re-key Key Pair and Derive Key cannot be carried out on an MDO key object. An attempt to do so will return an appropriate error as specified in the Error Handling section of **[KMIP-Spec]**.

## 3.38 Cryptographic Service

KMIP supports creation and registration of managed objects and retrieval of managed objects in both plaintext and optionally wrapped with another managed object. KMIP also includes support for a subset of the operations necessary for certificate management (certifying certificate requests and validating certificate hierarchies). KMIP defines a range of Hash-based and MAC-based key derivation options.

There are certain situations in which having capability for a KMIP client to request cryptographic operations from a KMIP server is beneficial in terms of simplifying the client implementation, strengthening the integration between the key management and cryptographic operations, or improving the overall security of a solution.

KMIP 1.2 adds support for cryptographic services in the form of client-to-server operations for cryptographic services using managed objects for encryption, decryption, signature generation, signature verification, MAC generation, MAC verification, random number generation, and general hashing.

This support for cryptographic services is similar to the approach taken in KMIP for certificates. The protocol supports a base set of operations on certificates that enable a key manager to act as a proxy for a Certification Authority or in fact operate as a Certification Authority in the contexts where that is appropriate. A KMIP server supporting cryptographic services may be acting as a proxy for another cryptographic device or in fact operating as a cryptographic device in the contexts where that is appropriate.

KMIP clients and KMIP servers using cryptographic services operations should be mindful of selecting a level of protection for the communication channel (the TLS connection) that provides sufficient protection of the plaintext data included in cryptographic operations and commensurate with the security strength of the operation. There is no requirement for the KMIP server to enforce selection of a level of protection.

Similarly, server policy regarding accepting random from a client (see section 2.5 regarding server policy) should reflect the level of confidence that that server has in a particular client or all clients. Issues in the quality or integrity of random provided in RNG Seed can affect key creation, nonce and IV generation, client-server TLS session key creation, and the random delivered to clients with the RNG Retrieve Operation. KMIP, as a protocol, does not itself enforce restrictions on the quality or nature of the random provided by a client in the RNG Seed operation.

A KMIP server that supports the RNG Retrieve and RNG Seed operations may have a single RNG for the server, an RNG which is shared in an unspecified manner by KMIP clients or a separate RNG for each KMIP client. There is no requirement for the KMIP server to implement any specific RNG model.

## 3.39 Passing Attestation Data

In some scenarios the server may want assurance of the integrity of the client's system before honoring a client's request. Additionally, the server may want a guarantee of the freshness of the attestation computation in the integrity measurement.

Generally, the process takes four passes:

1. The client sends a request to the server which requires attestation.
2. The server returns a random nonce to the client that will be used in the attestation computation to guarantee the freshness of the measurement.
3. The client sends a request to the server which includes the measurement of the client's system, and the measurement contains the nonce from the server.
4. The server verifies the measurement and sends the appropriate response to the client.

Passing attestation data with a client request can be achieved in KMIP as follows:

1. The client sends a request to the server with the Attestation Capable Indicator set to True in the request header.
2. If the request requires attestation, the server will return an "Attestation Required" error with a Nonce object in the response header. {If the client request fails for any reason other than "Attestation Required", the server will not include a nonce in the error message.}

  

3. The client uses the nonce received from the server in the attestation computation that will be used in the measurement.
    a. The client forms an Attestation Credential Object which contains either the measurement from the client or an assertion from a third party if the server is not capable or willing to verify the attestation data from the client.
    b. The client then issues a request which contains the Attestation Credential Object in the request header.
4. The server validates the measurement or assertion data in the Credential Object, checks that the nonce in the Credential Object matches one sent recently by the server, then sends the appropriate response to complete the request issued by the client. {If the measurement or assertion data in the Credential Object does not validate or if the nonce does not match one sent recently by the server, the server will return an "Attestation Failed" error instead of completing the request issued by the client.}

The server needs to be capable of processing and verifying multiple Credential Objects in the same request header since Attestation Credentials do not provide the same type of authentication as the Username and Password or Device Credential.

How frequently (e.g. every request, every 100 requests, etc.) the server generates a new random nonce depends on server policy. The lifetime of the nonce once the server has sent it to the client (i.e., the timeframe in which the client must return the nonce before needs to request a fresh nonce from the server) also depends on server policy.

If the client sends a request that requires attestation but the client has not set the Attestation Capable Indicator to True, then the server will send a "Permission Denied" error and will not include a Nonce object in the response header.

## 3.40 Split Key

KMIP v1.0 and KMIP v1.1 allow a client to register a Split Key that was created or otherwise obtained by the client, but offer no client operations to request a Split Key be generated or recombined by the server. The Create Split Key operation and Join Split Key operation are added to KMIP v1.2 to provide a more complete set of split key functionality.

To request the server generate a split key, the client sends a Create Split Key request that includes the Split Key parameters (Split Key Parts, Split Key Threshold, Split Key Method) and desired key attributes (e.g. Object Type, Cryptographic Length). If the client supplies the Unique Identifier of an existing base key in a Create Split Key request, the server will use the supplied key in the key splitting operation instead of generating a new one. The server will respond with a list of Unique Identifiers for the newly created Split Keys.

The client may want to add link attributes to more easily locate the complete set of related Split Keys as follows. The client adds a Previous Link from the Split Key with Key Part Identifier K to the Split Key with Key Part Identifier K-1 and a Next Link to the Split Key with Key Part Identifier K+1. Denoting the value of Split Key Parts by N, the client adds a Previous Link from the Split Key with Key Part Identifier 1 to the Split Key with Key Part Identifier N and a Next Link from the Split Key with Key Part Identifier N to the Split Key with Key Part Identifier 1. If the client supplies the Unique Identifier of an existing base key in a Create Split Key request, the client may want to add a Parent Link attribute from each newly generated Split Key to the base key that was supplied in the Create Split Key request.

To request the server recombine a set of split keys, the client sends a Join Split Key request that includes the type of object to be returned (e.g. Symmetric Key, Private Key, or Secret Data) and a list of Unique Identifiers of the Split Keys to be combined. The number of Unique Identifiers in the request needs to be at least the value of Split Key Threshold in the Split Keys to ensure the server will be able to combine the keys according to the Split Key Method. The server will respond with the Unique Identifier of the key obtained by combining the provided Split Keys.

## 3.41 Compromised Objects

A Cryptographic Object or Opaque Object may be compromised for a variety of reasons. In KMIP, a client indicates to the server that a Cryptographic Object is to be considered compromised by performing a Revoke Operation with a Revocation Reason of *Key Compromise* or *CA Compromise*. The KMIP client must provide a Compromise Occurrence Date (if the Revocation Reason is *Key Compromise*) and if it is unable to estimate when the compromise occurred then it should provide a Compromise Occurrence Date equal to the Initial Date.

The KMIP specification **[KMIP-Spec]** places no requirements on a KMIP server to perform any action on any Managed Object that references (i.e., via Link attributes) a Cryptographic Object or Opaque Object that a client has performed a Revoke operation with a Revocation Reason of *Key Compromise* or *CA Compromise*. However, KMIP users should be aware that there may be security relevant implications in continuing to use a Managed Cryptographic Object in the following circumstances:

- For a compromised Private Key, the linked Public Key and/or Certificate;
- For a compromised Public Key, the linked Private Key and/or Certificate;
- For a compromised Derived Key, the linked derived key and/or Secret Data Object

In these circumstances, it is the responsibility of the client to either check the state of the referenced Managed Object or to also perform a Revoke operation on the referenced Managed Object.

## 3.42 Elliptic Curve Cryptography (ECC) Algorithm Mapping

The KMIP Specification **[KMIP-Spec]** (see section 9.1.3.2.5) specifies a number of ECC algorithms (**[FIPS 186-4] [SEC2] [X9.62] [ECC-Brainpool] [RFC5639]**). These algorithms are defined in multiple source documents and in some cases, the same algorithm is known by multiple names since to the algorithm is defined in multiple documents. The following table provides a mapping of the ECC algorithms specified in the KMIP specification **[KMIP-Spec]**. The table identifies the

  

KMIP enumeration, the Object Identifier (OID) and multiples names (synonyms) for the ECC algorithms.

| Algorithm Name | KMIP Enumeration Value | OID | Algorithm Synonym(s) |
|---|---|---|---|
| P-192 | 00000001 | 1.2.840.10045.3.1.1 | SECP192R1<br>ANSIX9P192V1 |
| K-163 | 00000002 | 1.3.132.0.1 | SECT163K1 |
| B-163 | 00000003 | 1.3.132.0.15 | SECT163R2 |
| P-224 | 00000004 | 1.3.132.0.33 | SECP224R1 |
| K-233 | 00000005 | 1.3.132.0.26 | SECT233K1 |
| B-233 | 00000006 | 1.3.132.0.27 | SECT233R1 |
| P-256 | 00000007 | 1.2.840.10045.3.1.7 | SECP256R1<br>ANSIX9P256V1 |
| K-283 | 00000008 | 1.3.132.0.16 | SECT283K1 |
| B-283 | 00000009 | 1.3.132.0.17 | SECT283R1 |
| P-384 | 0000000A | 1.3.132.0.34 | SECP384R1 |
| K-409 | 0000000B | 1.3.132.0.36 | SECT409K1 |
| B-409 | 0000000C | 1.3.132.0.37 | SECT409R1 |
| P-521 | 0000000D | 1.3.132.0.35 | SECP521R1 |
| K-571 | 0000000E | 1.3.132.0.38 | SECT571K1 |
| B-571 | 0000000F | 1.3.132.0.39 | SECT571R1 |
| SECP112R1 | 00000010 | 1.3.132.0.6 | |
| SECP112R2 | 00000011 | 1.3.132.0.7 | |
| SECP128R1 | 00000012 | 1.3.132.0.28 | |
| SECP128R2 | 00000013 | 1.3.132.0.29 | |
| SECP160K1 | 00000014 | 1.3.132.0.9 | |
| SECP160R1 | 00000015 | 1.3.132.0.8 | |

  

| | | | |
|---|---|---|---|
| SECP160R2 | 00000016 | 1.3.132.0.30 | |
| SECP192K1 | 00000017 | 1.3.132.0.31 | |
| SECP192R1 | 00000001 | 1.2.840.10045.3.1.1 | P-192<br>ANSIX9P192V1 |
| SECP224K1 | 00000018 | 1.3.132.0.32 | |
| SECP224R1 | 00000004 | 1.3.132.0.33 | P-224 |
| SECP256K1 | 00000019 | 1.3.132.0.10 | |
| SECP256R1 | 00000007 | 1.2.840.10045.3.1.7 | P-256<br>ANSIX9P256V1 |
| SECP384R1 | 0000000A | 1.3.132.0.34 | P-384 |
| SECP521R1 | 0000000D | 1.3.132.0.35 | P-521 |
| SECT113R1 | 0000001A | 1.3.132.0.4 | |
| SECT113R2 | 0000001B | 1.3.132.0.5 | |
| SECT131R1 | 0000001C | 1.3.132.0.22 | |
| SECT131R2 | 0000001D | 1.3.132.0.23 | |
| SECT163K1 | 00000002 | 1.3.132.0.1 | K-163 |
| SECT163R1 | 0000001E | 1.3.132.0.2 | |
| SECT163R2 | 00000003 | 1.3.132.0.15 | B-163 |
| SECT193R1 | 0000001F | 1.3.132.0.24 | |
| SECT193R2 | 00000020 | 1.3.132.0.25 | |
| SECT233K1 | 00000005 | 1.3.132.0.26 | K-233 |
| SECT233R1 | 00000006 | 1.3.132.0.27 | B-233 |
| SECT239K1 | 00000021 | 1.3.132.0.3 | |
| SECT283K1 | 00000008 | 1.3.132.0.16 | K-283 |
| SECT283R1 | 00000009 | 1.3.132.0.17 | B-283 |
| SECT409K1 | 0000000B | 1.3.132.0.36 | K-409 |
| SECT409R1 | 0000000C | 1.3.132.0.37 | B-409 |

  

| | | | |
|---|---|---|---|
| SECT571K1 | 0000000E | 1.3.132.0.38 | K-571 |
| SECT571R1 | 0000000F | 1.3.132.0.39 | B-571 |
| ANSIX9P192V1 | 00000001 | 1.2.840.10045.3.1.1 | P-192<br>SECP192R1 |
| ANSIX9P192V2 | 00000022 | 1.2.840.10045.3.1.2 | |
| ANSIX9P192V3 | 00000023 | 1.2.840.10045.3.1.3 | |
| ANSIX9P239V1 | 00000024 | 1.2.840.10045.3.1.4 | |
| ANSIX9P239V2 | 00000025 | 1.2.840.10045.3.1.5 | |
| ANSIX9P239V3 | 00000026 | 1.2.840.10045.3.1.6 | |
| ANSIX9P256V1 | 00000007 | 1.2.840.10045.3.1.7 | P-256<br>SECP256R1 |
| ANSIX9C2PNB163V1 | 00000027 | 1.2.840.10045.3.0.1 | |
| ANSIX9C2PNB163V2 | 00000028 | 1.2.840.10045.3.0.2 | |
| ANSIX9C2PNB163V3 | 00000029 | 1.2.840.10045.3.0.3 | |
| ANSIX9C2PNB176V1 | 0000002A | 1.2.840.10045.3.0.4 | |
| ANSIX9C2TNB191V1 | 0000002B | 1.2.840.10045.3.0.5 | |
| ANSIX9C2TNB191V2 | 0000002C | 1.2.840.10045.3.0.6 | |
| ANSIX9C2TNB191V3 | 0000002D | 1.2.840.10045.3.0.7 | |
| ANSIX9C2PNB208W1 | 0000002E | 1.2.840.10045.3.0.10 | |
| ANSIX9C2TNB239V1 | 0000002F | 1.2.840.10045.3.0.11 | |
| ANSIX9C2TNB239V2 | 00000030 | 1.2.840.10045.3.0.12 | |
| ANSIX9C2TNB239V3 | 00000031 | 1.2.840.10045.3.0.13 | |
| ANSIX9C2PNB272W1 | 00000032 | 1.2.840.10045.3.0.16 | |
| ANSIX9C2PNB304W1 | 00000033 | 1.2.840.10045.3.0.17 | |
| ANSIX9C2TNB359V1 | 00000034 | 1.2.840.10045.3.0.18 | |
| ANSIX9C2PNB368W1 | 00000035 | 1.2.840.10045.3.0.19 | |
| ANSIX9C2TNB431R1 | 00000036 | 1.2.840.10045.3.0.20 | |

  

| | | | |
|---|---|---|---|
| BRAINPOOLP160R1 | 00000037 | 1.3.36.3.3.2.8.1.1.1 | |
| BRAINPOOLP160T1 | 00000038 | 1.3.36.3.3.2.8.1.1.2 | |
| BRAINPOOLP192R1 | 00000039 | 1.3.36.3.3.2.8.1.1.3 | |
| BRAINPOOLP192T1 | 0000003A | 1.3.36.3.3.2.8.1.1.4 | |
| BRAINPOOLP224R1 | 0000003B | 1.3.36.3.3.2.8.1.1.5 | |
| BRAINPOOLP224T1 | 0000003C | 1.3.36.3.3.2.8.1.1.6 | |
| BRAINPOOLP256R1 | 0000003D | 1.3.36.3.3.2.8.1.1.7 | |
| BRAINPOOLP256T1 | 0000003E | 1.3.36.3.3.2.8.1.1.8 | |
| BRAINPOOLP320R1 | 0000003F | 1.3.36.3.3.2.8.1.1.9 | |
| BRAINPOOLP320T1 | 00000040 | 1.3.36.3.3.2.8.1.1.10 | |
| BRAINPOOLP384R1 | 00000041 | 1.3.36.3.3.2.8.1.1.11 | |
| BRAINPOOLP384T1 | 00000042 | 1.3.36.3.3.2.8.1.1.12 | |
| BRAINPOOLP512R1 | 00000043 | 1.3.36.3.3.2.8.1.1.13 | |
| BRAINPOOLP512T1 | 00000044 | 1.3.36.3.3.2.8.1.1.14 | |

TABLE 3: ECC ALGORITHM MAPPING

  

# 4 Applying KMIP Functionality

This section describes how to apply the functionality described in the Key Management Interoperability Protocol Specification to address specific key management usage scenarios or to solve key management related issues.

## 4.1 Locate Queries

It is possible to formulate Locate queries to address any of the following conditions:

- Exact match of a transition to a given state. Locate the key(s) with a transition to a certain state at a specified time (t).
- Range match of a transition to a given state. Locate the key(s) with a transition to a certain state at any time at or between two specified times (t and t').
- Exact match of a state at a specified time. Locate the key(s) that are in a certain state at a specified time (t).
- Match of a state during an entire time range. Locate the key(s) that are in a certain state during an entire time specified with times (t and t'). Note that the Activation Date could occur at or before t and that the Deactivation Date could occur at or after t'+1.
- Match of a state at some point during a time range. Locate the key(s) that are in a certain state at some time at or between two specified times (t and t'). In this case, the transition to that state could be before the start of the specified time range.

This is accomplished by allowing any date/time attribute to be present either once (for an exact match) or at most twice (for a range match).

For instance, if the state we are interested in is Active, the Locate queries would be the following (corresponding to the bulleted list above):

- Exact match of a transition to a given state: Locate (ActivationDate(t)). Locate keys with an Activation Date of t.
- Range match of a transition to a given state: Locate (ActivationDate(t), ActivationDate(t')). Locate keys with an Activation Date at or between t and t'.
- Exact match of a state at a specified time: Locate (ActivationDate(0), ActivationDate(t), DeactivationDate(t+1), DeactivationDate(MAX_INT), CompromiseDate(t+1), CompromiseDate(MAX_INT) ). Locate keys in the Active state at time t, by looking for keys with a transition to Active before or until t, and a transition to Deactivated or Compromised after t (because we don't want the keys that have a transition to Deactivated or Compromised before t). The server assumes that keys without a DeactivationDate or CompromiseDate is equivalent to MAX_INT (i.e., infinite).
- Match of a state during an entire time range: Locate (ActivationDate(0), ActivationDate(t), DeactivationDate(t'+1), DeactivationDate(MAX_INT), CompromiseDate(t'+1), CompromiseDate(MAX_INT) ). Locate keys in the Active state during the entire time from t to t'.

  

- Match of a state at some point during a time range: Locate (ActivationDate(0), ActivationDate(t'-1), DeactivationDate(t+1), DeactivationDate(MAX_INT), CompromiseDate(t+1), CompromiseDate(MAX_INT)). Locate keys in the Active state at some time from t to t’, by looking for keys with a transition to Active between 0 and t’-1 and exit out of Active on or after t+1.

The queries would be similar for Initial Date, Deactivation Date, Compromise Date and Destroy Date.

In the case of the Destroyed-Compromise state, there are two dates recorded: the Destroy Date and the Compromise Date. For this state, the Locate operation would be expressed as follows:

- Exact match of a transition to a given state: Locate (CompromiseDate(t), State(Destroyed-Compromised)) and Locate (DestroyDate(t), State(Destroyed-Compromised)). KMIP does not support the OR in the Locate request, so two requests should be issued. Locate keys that were Destroyed and transitioned to the Destroyed-Compromised state at time t, and locate keys that were Compromised and transitioned to the Destroyed-Compromised state at time t.
- Range match of a transition to a given state: Locate (CompromiseDate(t), CompromiseDate(t'), State(Destroyed-Compromised)) and Locate (DestroyDate(t), DestroyDate(t'), State(Destroyed-Compromised)). Locate keys that are Destroyed-Compromised and were Compromised or Destroyed at or between t and t’.
- Exact match of a state at a specified time: Locate (CompromiseDate(0), CompromiseDate(t), DestroyDate(0), DestroyDate(t)); nothing else is needed, since there is no exit transition. Locate keys with a Compromise Date at or before t, and with a Destroy Date at or before t. These keys are, therefore, in the Destroyed-Compromised state at time t.
- Match of a state during an entire time range: Locate (CompromiseDate(0), CompromiseDate(t), DestroyDate(0), DestroyDate(t)). Same as above. As there is no exit transition from the Destroyed-Compromised state, the end of the range (t’) is irrelevant.
- Match of a state at some point during a time range: Locate (CompromiseDate(0), CompromiseDate(t'-1), DestroyDate(0), DestroyDate(t'-1)). Locate keys with a Compromise Date at or before t’-1, and with a Destroy Date at or before t’-1. As there is no exit transition from the Destroyed-Compromised state, the start of the range (t) is irrelevant.

## 4.2 Using Wrapped Keys with KMIP

KMIP provides the option to register and get keys in wrapped format. Clients request the server to return a wrapped key by including the Key Wrapping Specification in the Get Request Payload. Similarly, clients register a wrapped key by including the Key Wrapping Data in the Register Request Payload. The Wrapping Method identifies the type of mechanism used to wrap the key, but does not identify the algorithm or block cipher mode. It is possible to determine these from the attributes set for the specified Encryption Key or MAC/Signing Key. If a key has multiple Cryptographic Parameters set, clients may include the applicable parameters in Key Wrapping Specification. If omitted, the server chooses the Cryptographic Parameter attribute with the lowest index.

  

The Key Value includes both the Key Material and, optionally, attributes of the key; these may be provided by the client in the Register Request Payload; the server only includes attributes when requested in the Key Wrapping Specification of the Get Request Payload. The Key Value may be encrypted, signed/MACed, or both encrypted and signed/MACed (and vice versa). In addition, clients have the option to request or import a wrapped Key Block according to standards, such as ANSI TR-31, or vendor-specific key wrapping methods.

It is important to note that if the Key Wrapping Specification is included in the Get Request Payload, the Key Value may not necessarily be encrypted. If the Wrapping Method is MAC/sign, the returned Key Value is in plaintext, and the Key Wrapping Data includes the MAC or Signature of the Key Value.

Prior to wrapping or unwrapping a key, the server should verify that the wrapping key is allowed to be used for the specified purpose. For example, if the Unique ID of a symmetric key is specified in the Key Wrapping Specification inside the Get request, the symmetric key should have the “Wrap Key” bit set in its Cryptographic Usage Mask. Similarly, if the client registers a signed key, the server should verify that the Signature Key, as specified by the client inside the Key Wrapping Data, has the “Verify” bit set in the Cryptographic Usage Mask. If the wrapping key is not permitted to be used for the requested purpose (e.g., when the Cryptographic Usage Mask is not set), the server should return the Operation Failed result status.

### 4.2.1 Encrypt-only Example with a Symmetric Key as an Encryption Key for a Get Request and Response

The client sends a Get request to obtain a key that is stored on the server. When the client sends a Get request to the server, a Key Wrapping Specification may be included. If a Key Wrapping Specification is included in the Get request, and a client wants the requested key and its Cryptographic Usage Mask attribute to be wrapped with AES key wrap, the client includes the following information in the Key Wrapping Specification:

- Wrapping Method: Encrypt
- Encryption Key Information
  - Unique Key ID: Key ID of the AES wrapping key
  - Cryptographic Parameters: The Block Cipher Mode is NISTKeyWrap (not necessary if default block cipher mode for wrapping key is NISTKeyWrap)
- Attribute Name: Cryptographic Usage Mask

The server uses the Unique Key ID specified by the client to determine the attributes set for the proposed wrapping key. For example, the algorithm of the wrapping key is not explicitly specified inside the Key Wrapping Specification. The server determines the algorithm to be used for wrapping the key by identifying the Algorithm attribute set for the specified Encryption Key.

The Cryptographic Parameters attribute should be specified by the client if multiple instances of the Cryptographic Parameters exist, and the lowest index does not correspond to the NIST key wrap mode of operation. The server should verify that the AES wrapping key has NISTKeyWrap set as an allowable Block Cipher Mode, and that the “Wrap Key” bit is set in the Cryptographic Usage Mask.

If the correct data was provided to the server, and no conflicts exist, the server AES key wraps the Key Value (both the Key Material and the Cryptographic Usage Mask attribute) for the requested key with the wrapping key specified in the Encryption Key Information. The wrapped key (byte string) is returned in the server’s response inside the Key Value of the Key Block.

The Key Wrapping Data of the Key Block in the Get Response Payload includes the same data as specified in the Key Wrapping Specification of the Get Request Payload except for the Attribute Name.

### 4.2.2 Encrypt-only Example with a Symmetric Key as an Encryption Key for a Register Request and Response

The client sends a Register request to the server and includes the wrapped key and the Unique ID of the wrapping key inside the Request Payload. The wrapped key is provided to the server inside the Key Block. The Key Block includes the Key Value Type, the Key Value, and the Key Wrapping Data. The Key Value Type identifies the format of the Key Material, the Key Value consists of the Key Material and optional attributes that may be included to cryptographically bind the attributes to the Key Material, and the Key Wrapping Data identifies the wrapping mechanism and the encryption key used to wrap the object and the wrapping mechanism.

Similar to the example in 4.2.1 the key is wrapped using the AES key wrap. The Key Value includes four attributes: Cryptographic Algorithm, Cryptographic Length, Cryptographic Parameters, and Cryptographic Usage Mask.

The Key Wrapping Data includes the following information:

- Wrapping Method: Encrypt
- Encryption Key Information
    - Unique Key ID: Key ID of the AES wrapping key
    - Cryptographic Parameters: The Block Cipher Mode is NISTKeyWrap (not necessary if default block cipher mode for wrapping key is NISTKeyWrap)

Attributes do not need to be specified in the Key Wrapping Data. When registering a wrapped Key Value with attributes, clients may include these attributes inside the Key Value without specifying them inside the Template-Attribute.

Prior to unwrapping the key, the server determines the wrapping algorithm from the Algorithm attribute set for the specified Unique ID in the Encryption Key Information. The server verifies that the wrapping key may be used for the specified purpose. In particular, if the client includes the Cryptographic Parameters in the Encryption Key Information, the server verifies that the specified Block Cipher Mode is set for the wrapping key. The server also verifies that the wrapping key has the “Unwrap Key” bit set in the Cryptographic Usage Mask.

The Register Response Payload includes the Unique ID of the newly registered key and an optional list of attributes that were implicitly set by the server.

  

### 4.2.3 Encrypt-only Example with an Asymmetric Key as an Encryption Key for a Get Request and Response

The client sends a Get request to obtain a key (either symmetric or asymmetric) that is stored on the server. When the client sends a Get request to the server, a Key Wrapping Specification may be included. If a Key Wrapping Specification is included, and the key is to be wrapped with an RSA public key using the OAEP encryption scheme, the client includes the following information in the Key Wrapping Specification. Note that for this example, attributes for the requested key are not requested.

- Wrapping Method: Encrypt
- Encryption Key Information
  - Unique Key ID: Key ID of the RSA public key
  - Cryptographic Parameters:

    Padding Method: OAEP
    Hashing Algorithm: SHA-256

The Cryptographic Parameters attribute is specified by the client if multiple instances of Cryptographic Parameters exist for the wrapping key, and the lowest index does not correspond to the associated padding method. The server should verify that the specified Cryptographic Parameters in the Key Wrapping Specification and the “Wrap Key” bit in the Cryptographic Usage Mask are set for the corresponding wrapping key.

The Key Wrapping Data returned by the server in the Key Block of the Get Response Payload includes the same data as specified in the Key Wrapping Specification of the Get Request Payload.

For both OAEP and PSS, KMIP assumes that the Hashing Algorithm specified in the Cryptographic Parameters of the Get request is used for both the Mask Generation Function (MGF) and hashing data. The example above requires the server to use SHA-256 for both purposes.

### 4.2.4 MAC-only Example with an HMAC Key as an Authentication Key for a Get Request and Response

The client sends a Get request to obtain a key that is stored on the server. When the client sends a Get request to the server, a Key Wrapping Specification may be included. If a key and Custom Attribute (i.e., x-Nonce) is to be MACed with HMAC SHA-256, the following Key Wrapping Specification is specified:

- Wrapping Method: MAC/sign
- MAC/Signature Key Information
  - Unique Key ID: Key ID of the MACing key (note that the algorithm associated with this key would be HMAC-SHA256)
- Attribute Name: x-Nonce

For HMAC, no Cryptographic Parameters need to be specified, since the algorithm, including the hash function, may be determined from the Algorithm attribute set for the specified MAC Key.

  

The server should verify that the HMAC key has the “MAC Generate” bit set in the Cryptographic Usage Mask. Note that an HMAC key does not require the “Wrap Key” bit to be set in the Cryptographic Usage Mask.

The server creates an HMAC value over the Key Value if the specified MACing key may be used for the specified purpose and no conflicts exist. The Key Value is returned in plaintext, and the Key Block includes the following Key Wrapping Data:

- Wrapping Method: MAC/sign
- MAC/Signature Key Information
- Unique Key ID: Key ID of the MACing key
- MAC/Signature: HMAC result of the Key Value

In the example, the custom attribute x-Nonce was included to help clients, who are relying on the proxy model, to detect replay attacks. End-clients, who communicate with the key management server, may not support TLS and may not be able to rely on the message protection mechanisms provided by a security protocol. An alternative approach for these clients would be to use the custom attribute to hold a random number, counter, nonce, date, or time. The custom attribute needs to be created before requesting the server to return a wrapped key and is recommended to be set if clients frequently wrap/sign the same key with the same wrapping/signing key.

### 4.2.5 Registering a Wrapped Key as an Opaque Cryptographic Object

Clients may want to register and store a wrapped key on the server without the server being able to unwrap the key (i.e., the wrapping key is not known to the server). Instead of storing the wrapped key as an opaque object, clients have the option to store the wrapped key inside the Key Block as an opaque cryptographic object, i.e., the wrapped key is registered as a managed cryptographic object, but the encoding of the key is unknown to the server. Registering an opaque cryptographic object allows clients to set all the applicable attributes that apply to cryptographic objects (e.g., Cryptographic Algorithm and Cryptographic Length),

Opaque cryptographic objects are set by specifying the following inside the Key Block structure:

- Key Format Type: Opaque
- Key Material: Wrapped key as a Byte String

The Key Wrapping Data does not need to be specified.

### 4.2.6 Encoding Option for Wrapped Keys

KMIP provides the option to specify the Encoding Option inside the Key Wrapping Specification and Key Wrapping Data. This option allows users to Get or Register the Key Value in a non-TTLV encoded format. This may be desirable in a proxy environment, where the end-client is not KMIP-aware.

The Encoding Option is only available if no attributes are specified inside the Key Value. The server returns the Encoding Option Error if both the Encoding Option and Attribute Names are specified inside the Key Wrapping Specification. Similarly, the server is expected to return the

  

Encoding Option Error when registering a wrapped object with attributes inside the Key Value and the Encoding Option is set in the Key Wrapping Data. If no Encoding Option is specified, KMIP assumes that the Key Value is TTLV-encoded. Thus, by default, the complete TTLV-encoded Key Value content, as shown in the example below, is wrapped:

```
Key Material || Byte String   || Length   || Key Material Value
420043       || 08            || 00000010 || 0123456789ABCDEF0123456789ABCDEF
```

Some end-clients may not understand or have the space for anything more than the actual key material (i.e., 0123456789ABCDEF0123456789ABCDEF in the above example). To wrap only the Key Material value during a Get operation, the Encoding Option (00001 for no encoding) should be specified inside the Key Wrapping Specification. The same Encoding Option should be specified in the Key Wrapping Data when returning the non-TTLV encoded wrapped object inside the Get Response Payload or when registering a wrapped object in non-TTLV encoded format.

It is important to be aware of the risks involved when excluding the attributes from the Key Value. Binding the attributes to the key material in certain environments is essential to the security of the end-client. An untrusted proxy could change the attributes (provided separately via the Get Attributes operation) that determine how the key is being used (e.g., Cryptographic Usage). Including the attributes inside the Key Value and cryptographically binding it to the Key Material could prevent potential misuse of the cryptographic object and may prevent a replay attack if, for example, a nonce is included as a custom attribute. The exclusion of attributes and therefore the usage of the Encoding Option are only recommended in at least one of the following scenarios:

1. End-clients are registered with the KMIP server and are communicating with the server directly (i.e., the TLS connection is between the server and client).
2. The environment is controlled and non-KMIP-aware end-clients are aware how wrapped cryptographic objects (possibly Raw keys) from the KMIP server should be used without having to rely on the attributes provided by the Get Attributes operation.
3. The wrapped cryptographic object consists of attributes inside the Key Material value. These attributes are not interpreted by the KMIP server, but are understood by the end-client. This may be the case if the Key Format Type is opaque or vendor-specific.
4. The proxy communicating with the KMIP server on behalf of the end-client is considered to be trusted and is operating in a secure environment.

Registering a wrapped object without attributes is not recommended in a proxy environment, unless scenario 4 is met.

## 4.3 Interoperable Key Naming for Tape

This section describes methods and provides examples for creating and storing key identifiers that are interoperable across multi-vendor KMIP clients, using the KMIP Tape Library Profile Version 1.0.

  

### 4.3.1 Native Tape Encryption by a KMIP Client

A common method for naming and retrieving keys is needed to support moving tape cartridges between 2 or more KMIP-compliant tape libraries that are all registered with the same KMIP key manager.

#### 4.3.1.1 Method Overview

The method uses the KMIP Tape Library Profile. This profile specifies use of the KMIP Application Specific Information (ASI) attribute. The method supports both client-generated and server-generated key identifiers.

The key identifier is a KMIP string, composed of hexadecimal numeric characters. This string of characters is unique within a chosen namespace. Methods of generating the string are determined by policy. The LIBRARY-LTO namespace is preferred for maximum interoperability.

A compressed (numeric) transformation of the identifier string is stored in the tape format's Key Associated Data. This allows for future retrieval of the key for decryption.

Interoperability is achieved by a) standardized algorithms to map byte values between the numeric (KAD) and text (ASI) representations of the identifier; and b) standardized ordering of bytes within the KAD so the identifier can be re-assembled in the correct sequence by other compliant implementations. Examples of the algorithms are provided below.

#### 4.3.1.2 Definitions

*Key Associated Data (KAD):* Part of the tape format. May be segmented into authenticated and unauthenticated fields. KAD usage is detailed in the SCSI SSC-3 standard from the T10 organization.

*Application Specific Information (ASI):* A KMIP attribute.

*Hexadecimal numeric characters*: Case-sensitive, printable, single byte ASCII characters representing the numbers 0 through 9 and uppercase alpha A through F. (US-ASCII characters 30h-39h and 41h-46h).

Hexadecimal numeric characters are always paired, each pair representing a single 8-bit numeric value. A leading zero character is provided, if necessary, so that every byte in the tape's KAD is represented by exactly 2 hexadecimal numeric characters.

*N(k):* The number of bytes in the tape format combined KAD fields (both authenticated and unauthenticated).

*N(a), N(u):* The number of bytes in the tape formats authenticated, and unauthenticated KAD fields, respectively.

#### 4.3.1.3 Implementation Example of Algorithm 1. Key identifier string to numeric direction (Converting the ASI string to tape format's KAD)

Refer to the KMIP Tape profile for algorithm 1.

This algorithm is associated with writing the KAD, typically to allow future retrieval of a key. An example implementation is as follows.

1. The client creates a key identifier or obtains one from the server. The identifier is a KMIP string of hexadecimal numeric characters. Copy the string to an input buffer of size 2*N(k) bytes. For LTO4, an 88 character string is sufficient to represent any key name stored directly in the KAD fields. For LTO5, a 184 character string is sufficient to represent any key name stored directly in the KAD fields.
2. Define output buffers for unauthenticated KAD, and authenticated KAD, of size N(u) and N(a) respectively. For LTO4, this would be 32 bytes of unauthenticated data, and 12 bytes of authenticated data. For LTO5, this would be 32 bytes of unauthenticated data and 60 bytes of authenticated data.
3. Define the standard POSIX (also known as C) locale. Each character in the string is a single-byte US-ASCII character.
4. First, populate the authenticated KAD buffer, converting a sub-string consisting of the last (rightmost) 2*N(a) characters of the key identifier string.
5. When the authenticated KAD is filled, next populate the unauthenticated KAD buffer, by converting the remaining hexadecimal character pairs (if any) of the identifier string.

#### 4.3.1.4 Implementation Example of Algorithm 2. Numeric to key identifier string direction (Converting tape format's KAD to ASI string)

This algorithm is associated with reading the KAD, typically in preparation for retrieving a key. An example implementation is as follows

1. Define an input buffer sized for N(k). For LTO4, N(k) is 44 bytes (12 bytes authenticated, 32 unauthenticated). For LTO5, N(k) is 92 bytes (60 bytes authenticated, 32 bytes unauthenticated).
2. Define an output buffer sufficient to contain a string with a maximum length of 2*N(k) bytes.
3. Define the standard POSIX (also known as C) locale. Each character in the string is a single-byte US-ASCII character.
4. First, copy the tape format's unauthenticated KAD data (if any) to the input buffer. Next, bytes from the authenticated KAD are concatenated, after the unauthenticated bytes. In many implementations the unauthenticated KAD is empty, and in those cases the entire input buffer will be populated with bytes from authenticated KAD.
5. For each byte in the input buffer, convert to US-ASCII as follows:
6. Convert the byte's value to exactly 2 hexadecimal numeric characters, including a leading 0 where necessary. Append these 2 numeric characters to the output buffer, with the high-nibble represented by the left-most hexadecimal numeric character.

#### 4.3.1.5 Usage Example

The following usage example will create a key identifier which can be stored in ASI. The identifier will then be translated for storage into a tape format's KAD, using algorithm 1. Both LTO4 and LTO5 examples of KAD contents are provided.

  

The reverse translation from KAD bytes to the KMIP key identifier is not shown, but would be accomplished via algorithm 2. This re-constructed key identifier string would be used to Locate the key via ASI.

**Example of creating a key identifier.** Implementation-specific material is used to generate a key identifier. The content of this material is based on server or client policy. An example of a text string which could be used to generate a KMIP key identifier for tape is as follows.

`SN12345`**`6`**`_MFR:XY`**`Z`**` INC_BA`**`R`**`12345_T`**`M`**`2013123`**`4`**

This example is a set of 40 characters which will be used to create a KMIP key identifier for use as specified in the KMIP Tape Profile. Every 8th character is bold.

This set of characters is suitable as a key identifier for either LTO4 or LTO5, since it will fit within the smaller 44 character KAD space of LTO4.

The corresponding KMIP key identifier, which is a string of hexadecimal numeric character pairs, is shown below. This string will be stored in ASI Application Data.

53 4E 31 32 33 34 35 **36** 5F 4D 46 52 3A 58 59 **5A** 20 49 4E 43 5F 42 41 **52**

31 32 33 34 35 5F 54 **4D** 32 30 31 33 31 32 33 **34**

Spaces are shown for to improve readability, but are NOT part of the ASI string. Every 8th hexadecimal numeric pair is bold.

Note the identifier has exactly 2x more characters than the material used to generate the KMIP key identifier.

**Translating the key identifier to KAD bytes (LTO4).** The corresponding KAD content, for use with an LTO4 tape cartridge is shown in the following figure.

  

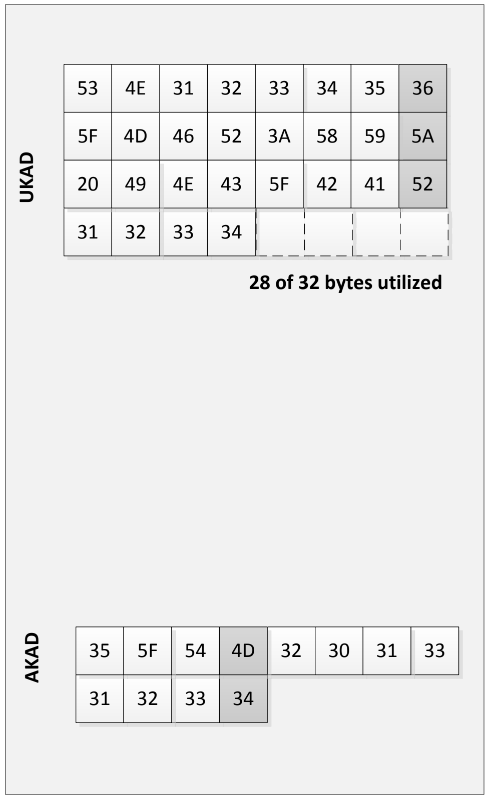


**FIGURE 2: KAD CONTENT FOR LTO4**

Each square is 1 byte (8 bits). The contents of each square is the 8 bit value which represents a pair of hexadecimal numeric characters in the KMIP key identifier string.

Every 8$^{th}$ byte of KAD is shaded.

The KAD was populated by converting the rightmost 24 characters (12 character pairs) of the identifier string into bytes of authenticated KAD. The remaining characters of the identifier were written to unauthenticated KAD.

**Translating the key identifier to KAD bytes (LTO5).** The corresponding KAD for use with an LTO5 and later tape cartridge is shown in the following figure.

  

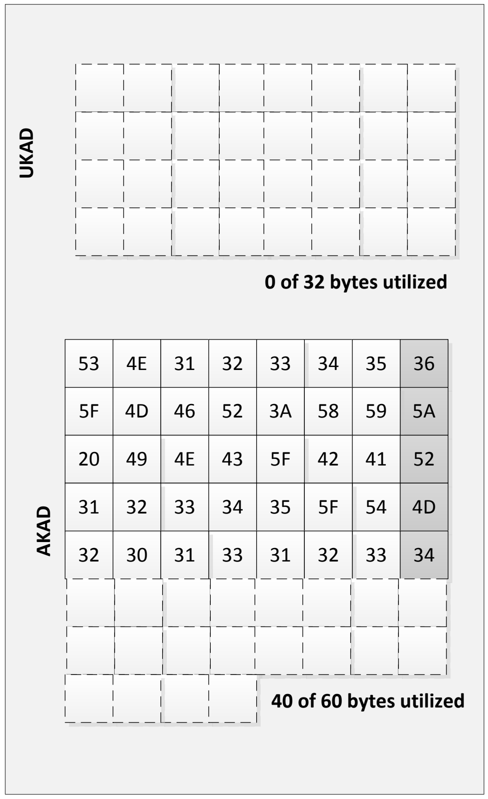


**Figure 3: KAD Content for LTO5**

Each square is 1 byte (8 bits). The contents of each square is the 8 bit value which represents a pair of hexadecimal numeric characters in the key identifier string.

Every 8th byte of KAD is shaded.

The KAD was populated by converting the rightmost 80 characters (40 character pairs) of the identifier string into bytes of authenticated KAD. The unauthenticated KAD is not used because all of the data fits within authenticated KAD.

## 4.4 Query Extension Information

The Extension Information structure added to KMIP 1.1 and the Query Extension List and Query Extension Map functions of the Query Operation provide a mechanism for a KMIP client to be able to determine which extensions a KMIP server supports.

A client may request the list of Extensions supported by a KMIP 1.1 server by specifying the Query Extension List value in the Query Function field. This provides the names of the supported extensions.

Example output:

Extension Information

Extension Name: ACME LOCATION

Extension Information

Extension Name: ACME ZIP CODE

A client may request the details of Extensions supported by a KMIP 1.1 server by specifying the Query Extension Map value in the Query Function field. This provides the names of the supported extensions.

Example output:

Extension Information

Extension Name: ACME LOCATION

Extension Tag: 0x54AA01

Extension Type: Text String

Extension Information

Extension Name: ACME ZIP CODE

Extension Tag: 0x54AA02

Extension Type: Integer

## 4.5 Registering Extension Information

As tag values and their interpretation for the most part should be known for a client and server to meaningfully use an extension, the following registration procedure should be used.

1. Document the Extensions including:
    a. Extension Tag, Extension Name, Extension Type values to be reserved
    b. A brief description of the purpose of the Extension
    c. Example use case messages (requests and responses)
    d. Example Guidance
2. Send the Document to the KMIP TC requesting review
3. Request a KMIP TC ballot on accepting the reservation of the Extension

It is anticipated that a template document may be produced for this registration process.

## 4.6 Using KMIP for PGP Keys

PGP, both as vendor product and as standard, provides a rich environment for key management that addresses significant use cases related to such areas as secure exchange of email, documents and other resources. Although KMIP is by no means required for support of PGP environments, it can provide a valuable mechanism for movement of PGP keys between a particular PGP environment, such as Symantec Encryption Management Server (SEMS, née PGP Universal), and another key management environment.

  

KMIP does not attempt to represent the full range of functionality in PGP environments. However, the use cases related to movement of PGP keys across environments, described in the KMIP Use Cases document, can be supported by taking advantage both of the PGP-specific capabilities in KMIP, such as the PGP Key object introduced in KMIP V1.2, and of KMIP messages, objects, operations and attributes in general.

In order to support the PGP use cases, KMIP V1.2 introduces new capabilities:

- PGP Key managed object
- Alternative Name attribute
- Enhancements to Link attribute

The PGP Key managed object contains a PGP key (specified in **[RFC4880]**) as an opaque blob. KMIP compliant servers do not need to understand the fine structure of PGP keys. The intention here is that PGP-enabled clients be able to discover the PGP Key managed cryptographic objects by searching for one of the various names contained within the block. The Alternative Name attribute can be used to specify one or more names (e.g. User IDs) that are attached to the PGP Key object. The PGP-enabled clients are expected to digest the PGP Key object and properly assign these Alternative Name attributes on to the cryptographic managed object. The KMIP server does not have to do this work.

Internally, PGP keys may contain many public-private key pairs, each tied to a specific type of encryption operations (one key for signing, one for encryption, and one to tie the other two together in a trust relationship is one typical arrangement.) The Link attribute supports new values that enable the description of this set of PGP Key relationships. The new values are parent, child, previous and next. For example, the private and public keys associated with a PGP Key can be pointed to from the PGP Key with the “child” link attribute. Additional Decryption Keys (ADK) can be pointed to from the PGP Key with the “child” link attribute and can be point to each other with the “previous” and “next” link attributes. In this way, the link attributes can be used to define the structural relationships required to establish the web of trust for a PGP Key.

As mentioned above, KMIP does not attempt to represent all the information about PGP keys that would be managed within a PGP implementation. For example, policies such as algorithms supported, by a PGP key are not expressed within KMIP. Instead, KMIP enables the specification of these attributes, if necessary, as information enclosed within the opaque value defined for a given PGP key. This information would be handled by security administration and out-of-band coordination between the PGP environments that participate in the KMIP exchanges related to PGP keys.

KMIP complaint servers are not expected to be able to create PGP Key objects from scratch. PGP-enabled clients will do the key creation and pass the resulting information up to KMIP.

## 4.7 KMIP Client Registration Models

There are several common approaches to registering KMIP clients with KMIP servers:

- Manual client registration within a single trust boundary

  

- Automatic client registration across multiple trust boundaries
- Configuring a KMIP Server for use with Automatic Client Registration

The goal of these approaches is to establish the KMIP-interoperable secure channel or channels between KMIP servers and clients, such as a mutually-authenticated TLS channel.

In order to support the goal of establishing an interoperable approach to establishing this channel, this section provides more detailed information about these approaches to client registration.

Reflecting common usage for KMIP, all three of the scenarios described below discuss the use of X.509 certificates for trust establishment; other mechanisms, such as quantum key distribution, may be used instead but are not described here. Similarly, all three scenarios describe the establishment of a mutually-authenticated TLS connection as the basis trusted exchange of KMIP messages, corresponding to the published KMIP authentication suite profiles; other authentication mechanisms can be used with KMIP, but are not described here.

### 4.7.1 Manual Client Registration

In this approach, there is no assumption of pre-population of authentication credentials in the client, such as by installing an X.509 certificate into a tape library or drive during the manufacturing process. Rather, a credential is propagated out-of-band to the client administrator, who installs it into the client environment. The credential is then used on initial and subsequent contact between the client and server systems.

The most common registration model that takes this approach entails the server administrator creating a package that contains 1) X.509 certificate that the client will use to identify itself to the server when creating a TLS mutually-authenticated session; 2) information about the X.509 certificate that will be presented by the server to the client during negotiation of the mutual authentication, enabling the client to verify the server identity; and 3) possibly additional information that can be included in the credential of the KMIP message sent across the established channel, such as to provide finer granularity for particular drives within a tape library. As indicated, the use of this package of materials takes place during two phases: first during the establishment of the TLS secure channel; second during the transmission of KMIP messages. The server administrator must have configured the server to recognize the X.509 certificate presented by the client, to present the correct X.509 certificate of its own to the client in return and to recognize the additional information provided in the credential object in the KMIP message, if any.

In this model, KMIP is not used to transmit the X.509 certificate and server information used in establishing the secure channel. There is nothing to prevent KMIP being used to send this information; but commonly this is done using mechanisms other than KMIP, nor is there any expectation that KMIP is a required or default mechanism for propagating the credential and the information. The distribution mechanism, therefore, may well vary across vendors.

  

The use of additional information as the credential in the KMIP message is also neither required nor a default. Inclusion of such a credential in the package distributed to the client administrator and in one or more KMIP messages is also, therefore, likely to vary across vendors.

### 4.7.2 Automated Client Registration

In this approach the credential used to establish a mutually-authenticated TLS connection is not provided in the package provided by the server administrator. Instead, the establishment of trust between the client and server is accomplished by some other mechanism. In one common version of this approach, an X.509 certificate is installed in a client device during the manufacturing process. This certificate is then used as a bootstrap mechanism for the subsequent exchange of the kind of information exchanged between client administrator and server administrator in section 4.7.1.

There will be typically be configuration activity for the client device based on information, such as a Service ID, received from the server administrator. Once the client administrator initiates auto-registration, the client device sends the X.509 certificate to the server, for example in order to use it to establish an initial TLS session. The server then sends the equivalent of the registration packet in section 4.7.1 above to the client and the client returns the certificate to be used for establishing the secure TLS channel with the server.

In this model, one common variant is to require administrator intervention to determine whether the initial client certificate should be accepted. The scenario above assumes that the return of the server's packet of registration is immediate and automatic; alternatively, the return of the packet of information may be done manually by the server administrator, as in section 4.7.1 above; or the return of the packet of server information may be done by the server, but only after that action has been approved by an administrator.

As discussed in section 4.7.1, KMIP can be used by the client in sending the X.509 certificates to the server. However, this is not required and is currently not typical. If it is sent to the server using a KMIP register operation, the server must be able to distinguish that this operation is intended not only to register the cryptographic object, but also to initiate the registration of the client as a legitimate participant in KMIP message exchange.

### 4.7.3 Registering Sub-Clients Based on a Trusted Primary Client

A third common model for registering sub-clients of a trusted client. In this model, the establishment of trust between the client and server can be accomplished using either of the approaches in section 4.7.1 or 4.7.2. However, the server may also send additional information to the client, such as a "tenant identifier", which it will have to provide to sub-clients for them to use they attempt to register individually. The individual sub-clients would follow a registration model such as that described in section 4.7.2, but would also provide the tenant identifier along with the X.509 certificate so that the server can decide whether to accept the client, based on such criteria as the TCP/IP address of the sub-client relative to that of the primary client.

  

This approach is common for tiered clients such as virtual machines that need to be grouped based on their association with a larger trusted entity, but that also need individual identities and trust relationships established based on those identities.

KMIP can be used for sending both the client certificate and the tenant identifier to the server. But again this is no currently common practice.

  

# 5 Deprecated KMIP Functionality

This section describes KMIP functionality that has been deprecated.

Use of deprecated functionality is discouraged since such functionality may be dropped in a future release of the **[KMIP-Spec]**.

## 5.1 KMIP Deprecation Rule

Items in the normative KMIP Specification **[KMIP-Spec]** document can be marked deprecated in any document version, but will be removed only in a major version. Similarly, conformance clauses or other normative information in the KMIP Profiles **[KMIP-Prof]** document can be deprecated in any document version, but removed only in a major version. Information in the non-normative KMIP Usage Guide [this document] and KMIP Test Cases **[KMIP-TC]** documents may be removed in any document version.

## 5.2 Certificate Attribute Related Fields

The KMIP v1.0 *Certificate Identifier, Certificate Subject* and *Certificate Issuer* attributes are populated from values found within X.509 public key or PGP certificates. In KMIP v1.0 these fields were encoded as *Text String*, but the values of these fields are obtained from certificates which are *ASN.1 (X.509) or octet (PGP)* encoded. In KMIP v1.1, the data type associated with these fields was changed from *Text String* to *Byte String* so that the values of these fields parsed from the certificates can be preserved and no conversion from the encoded values into a text string is necessary.

Since these certificate-related attributes and associated fields were included as part of the v1.0 KMIP specification and that there may be implementations supporting these attributes using the Text String encoding, a decision was made to deprecate these attributes in KMIP v1.1 and replace them with newly named attributes and fields. As part of this change, separate certificate-related attributes for X.509 certificates were introduced.

Table 4 provides a list of the deprecated certificate-related attributes and fields along with their corresponding tag value.

| Deprecated Attribute/Field | Deprecated Tag Value |
|---|---|
| Certificate Identifier | 420014 |
| Certificate Issuer | 420015 |
| Certificate Issuer Alternative Name | 420016 |
| Certificate Issuer Distinguished Name | 420017 |
| Certificate Subject | 42001A |

  

| | |
|---|---|
| Certificate Subject Alternative Name | 42001B |
| Certificate Subject Distinguished Name | 42001C |
| Issuer | 42003B |
| Serial Number | 420087 |

**Table 4: Deprecated Certificate Related Attributes and Fields**

Table 5 provides a mapping of v1.0 to v1.1 certificate attributes and fields.

| Deprecated V1.0 Attribute | Deprecated V1.0 Field | New V1.1 Attribute | New V1.1 Field |
|---|---|---|---|
| Certificate Identifier | Issuer | X.509 Certificate Identifier | Issuer Distinguished Name |
| | Serial Number | | Certificate Serial Number |
| Certificate Issuer | Certificate Issuer Distinguished Name | X.509 Certificate Issuer | Issuer Distinguished Name |
| | Certificate Issuer Alternative Name | | Issuer Alternative Name |
| Certificate Subject | Certificate Subject Distinguished Name | X.509 Certificate Subject | Subject Distinguished Name |
| | Certificate Subject Alternative Name | | Subject Alternative Name |

**Table 5: Mapping of v1.0 to v1.1 Certificate Related Attributes and Fields**

## 5.3 PGP Certificate and Certificate Request Types

KMIP 1.0 and 1.1 included support for PGP via a PGP Certificate Type and associated PGP Certificate Request Type. However the certificate concept, which is typically associated with X.509 public key certificates, is not well suited for describing PGP keys and associated credentials as specified in **[RFC4880]**. For example, PGP may associate multiple asymmetric key pairs and associated public key certificates to the same subject, while a X.509 certificate associates a single public key to a subject. As a result of these differences it was difficult to apply the X.509 public key certificate structure and attributes to PGP credentials in a meaningful way.

KMIP 1.2 introduces changes and additions to KMIP that allow PGP usage scenarios as specified in **[RFC4880]** to be better supported within KMIP. (See Section 4.6 for more information.) These changes include the deprecation of the PGP Certificate Type and PGP Certificate Request Type concepts and the introduction of a new PGP Key managed cryptographic object.

Table 6 lists the PGP Certificate Type enumeration which has been deprecated as of KMIP 1.2.

  

| Certificate Type | |
|---|---|
| **Name** | **Value** |
| PGP | `00000002 (deprecated)` |

**TABLE 6: DEPRECATED PGP CERTIFICATE TYPE**

Table 7 lists the PGP Certificate Request Type enumeration which has been deprecated as of KMIP 1.2

| Certificate Request Type | |
|---|---|
| **Name** | **Value** |
| PGP | `00000004 (deprecated)` |

**TABLE 7: DEPRECATED PGP-CERTIFICATE REQUEST TYPE**

  

# 6 Implementation Conformance

This document is intended to be informational only and as such has no conformance clauses. The conformance requirements for the KMIP Specification can be found in the "KMIP Specification" document itself, at the URL noted in the “Normative References” section of this document.

  

# Appendix A. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

## Participants in KMIP Usage Guide V1.2

Hal Aldridge, Sypris Electronics
Mike Allen, Symantec
Gordon Arnold, IBM
Todd Arnold, IBM
Richard Austin, Hewlett-Packard
Lars Bagnert, PrimeKey
Elaine Barker, NIST
Peter Bartok, Venafi, Inc.
Tom Benjamin, IBM
Anthony Berglas, Cryptsoft
Mathias Björkqvist, IBM
Kevin Bocket, Venafi
Anne Bolgert, IBM
Alan Brown, Thales e-Security
Tim Bruce, CA Technologies
Chris Burchett, Credant Technologies, Inc.
Kelley Burgin, National Security Agency
Robert Burns, Thales e-Security
Chuck Castleton, Venafi
Kenli Chong, QuintessenceLabs
John Clark, Hewlett-Packard
Tom Clifford, Symantec Corp.
Tony Cox, Cryptsoft
Russell Dietz, SafeNet, Inc
Graydon Dodson, Lexmark International Inc.
Vinod Duggirala, EMC Corporation
Chris Dunn, SafeNet, Inc.
Michael Duren, Sypris Electronics
James Dzierzanowski, American Express CCoE
Faisal Faruqui, Thales e-Security
Stan Feather, Hewlett-Packard
David Finkelstein, Symantec Corp.
James Fitzgerald, SafeNet, Inc.
Indra Fitzgerald, Hewlett-Packard
Judith Furlong, EMC Corporation
Susan Gleeson, Oracle
Robert Griffin, EMC Corporation
Paul Grojean, Individual
Robert Haas, IBM
Thomas Hardjono, M.I.T.
ChengDong He, Huawei Technologies Co., Ltd.
Steve He, Vormetric
Kurt Heberlein, Hewlett-Packard
Larry Hofer, Emulex Corporation
Maryann Hondo, IBM
Walt Hubis, NetApp

  

Tim Hudson, Cryptsoft
Jonas Iggbom, Venafi, Inc.
Sitaram Inguva, American Express CCoE
Jay Jacobs, Target Corporation
Glen Jaquette, IBM
Mahadev Karadiguddi, NetApp
Greg Kazmierczak, Wave Systems Corp.
Marc Kenig, SafeNet, Inc.
Mark Knight, Thales e-Security
Kathy Kriese, Symantec Corporation
Mark Lambiase, SecureAuth
John Leiseboer, Quintenssence Labs
Hal Lockhart, Oracle Corporation
Robert Lockhart, Thales e-Security
Anne Luk, Cryptsoft
Sairam Manidi, Freescale
Luther Martin, Voltage Security
Neil McEvoy, iFOSSF
Marina Milshtein, Individual
Dale Moberg, Axway Software
Jishnu Mukeri, Hewlett-Packard
Bryan Olson, Hewlett-Packard
John Peck, IBM
Rob Philpott, EMC Corporation
Denis Pochuev, SafeNet, Inc.
Reid Poole, Venafi, Inc.
Ajai Puri, SafeNet, Inc.
Saravanan Ramalingam, Thales e-Security
Peter Reed, SafeNet, Inc.
Bruce Rich, IBM
Christina Richards, American Express CCoE
Warren Robbins, Dell
Peter Robinson, EMC Corporation
Scott Rotondo, Oracle
Saikat Saha, Oracle
Anil Saldhana, Red Hat
Subhash Sankuratripati, NetApp
Boris Schumperli, Cryptomathic
Greg Singh, QuintessenceLabs
David Smith, Venafi, Inc.
Brian Spector, Certivox
Terence Spies, Voltage Security
Deborah Steckroth, RouteOne LLC
Michael Stevens, QuintessenceLabs
Marcus Streets, Thales e-Security
Satish Sundar, IBM
Kiran Thota, VMware
Somanchi Trinath, Freescale Semiconductor, Inc.
Nathan Turajski, Thales e-Security
Sean Turner, IECA, Inc.
Paul Turner, Venafi, Inc.
Rod Wideman, Quantum Corporation
Steven Wierenga, Hewlett-Packard
Jin Wong, QuintessenceLabs
Sameer Yami, Thales e-Security
Peter Yee, EMC Corporation

  

Krishna Yellepeddy, IBM
Catherine Ying, SafeNet, Inc.
Tatu Ylonen, SSH Communications Security (Tectia Corp)
Michael Yoder, Vormetric. Inc.
Magda Zdunkiewicz, Cryptsoft
Peter Zelechoski, Election Systems & Software

  

# Appendix B. Acronyms

The following abbreviations and acronyms are used in this document:

3DES - Triple Data Encryption Standard

ADK - Additional Decryption Key

AES - Advanced Encryption Standard specified in **[FIPS 197]**

ANSI - American National Standards Institute

ARQC - Authorization Request Cryptogram

ASCII - American Standard Code for Information Interchange

ASI - Application Specific Information

ASN.1 - Abstract Syntax Notation One

CA - Certification Authority

CBC - Cipher Block Chaining specified in **[SP800-38A]**

CMC - Certificate Management Messages over CMS specified in **[RFC5272]**

CMP - Certificate Management Protocol specified in **[RFC4210]**

CRL - Certificate Revocation List specified in **[X.509]**

CRMF - Certificate Request Message Format specified in **[RFC4211]**

CVC - Card Verification Code

DEK - Data Encryption Key

DH - Diffie-Hellman specified in **[X9.42]**

DSA - Digital Signature Algorithm specified in **[FIPS 186-4]**

DSS - Digital Signature Standard

ECC - Elliptic Curve Cryptography

ECDH - Elliptic Curve Diffie Hellman

ECDSA - Elliptic Curve Digital Signature Algorithm

FIPS - Federal Information Processing Standard

GCM - Galois/Counter Mode specified in **[SP800-38D]**

HMAC - Keyed-Hash Message Authentication Code specified in **[FIPS 198-1]**

HSM - Hardware Security Module

HTTP - Hyper Text Transfer Protocol

HTTPS - Hyper Text Transfer Protocol (Secure socket)

ID - Identification

  

IP - Internet Protocol

IPSec - Internet Protocol Security

ITU - International Telecommunication Union

KAD - Key Associated Data

KEK - Key Encryption Key

KMIP - Key Management Interoperability Protocol

LTO4 - Linear Tape-Open, Generation 4

LTO5 - Linear Tape-Open, Generation 5

LTO6 - Linear Tape-Open, Generation 6

MAC - Message Authentication Code

MD5 - Message Digest 5 Algorithm specified in **[RFC1321]**

MDO - Meta-Data Only

MGF - Mask Generation Function

NIST - National Institute of Standards and Technology

OAEP - Optimal Asymmetric Encryption Padding specified in **[PKCS#1]**

OID - Object Identifier

PEM - Privacy Enhanced Mail specified in **[RFC1421]**

PGP - OpenPGP specified in **[RFC4880]**

PKCS - Public-Key Cryptography Standards

POP - Proof of Possession

POSIX - Portable Operating System Interface

PSS - Probabilistic Signature Scheme specified in **[PKCS#1]**

RNG - Random Number Generator

RSA - Rivest, Shamir, Adelman (an algorithm)

SEMS - Symantec Encryption Management Server

SHA - Secure Hash Algorithm specified in **[ECC-Brainpool]**

**ECC Brainpool Standard** *Curves and Curve Generation v. 1.0.19.10.2005*, http://www.ecc-brainpool.org/download/Domain-parameters.pdf.

**[FIPS 180-4]**

SP - Special Publication

SMIME - Secure Multipurpose Internet Mail Extensions

TCP - Transport Control Protocol

TDEA - Triple Data Encryption Algorithm

  

TLS - Transport Layer Security

TTLV - Tag, Type, Length, Value

URI - Uniform Resource Identifier

  

# Appendix C. Table of Figures and Tables

## Table of Figures



## Table of Tables



  

# Appendix D. KMIP Specification Cross Reference

| **Reference Term** | **KMIP 1.0** | **KMIP 1.1** | **KMIP 1.2** |
|---|---|---|---|
| **1 Introduction** | | | |
| *Non-Normative References* | 1.3. | 1.3. | 1.3. |
| *Normative References* | 1.2. | 1.2. | 1.2. |
| *Terminology* | 1.1. | 1.1. | 1.1. |
| **2 Objects** | | | |
| *Attribute* | 2.1.1. | 2.1.1. | 2.1.1. |
| *Base Objects* | 2.1. | 2.1. | 2.1. |
| *Certificate* | 2.2.1. | 2.2.1. | 2.2.1. |
| *Credential* | 2.1.2. | 2.1.2. | 2.1.2. |
| *Data* | - | - | 2.1.10. |
| *Data Length* | - | - | 2.1.11. |
| *Extension Information* | - | 2.1.9. | 2.1.9. |
| *Key Block* | 2.1.3. | 2.1.3. | 2.1.3. |
| *Key Value* | 2.1.4. | 2.1.4. | 2.1.4. |
| *Key Wrapping Data* | 2.1.5. | 2.1.5. | 2.1.5. |
| *Key Wrapping Specification* | 2.1.6. | 2.1.6. | 2.1.6. |
| *MAC Data* | - | - | 2.1.13. |
| *Managed Objects* | 2.2. | 2.2. | 2.2. |
| *Nonce* | - | - | 2.1.14. |
| *Opaque Object* | 2.2.8. | 2.2.8. | 2.2.8. |
| *PGP Key* | - | - | 2.2.9. |
| *Private Key* | 2.2.4. | 2.2.4. | 2.2.4. |
| *Public Key* | 2.2.3. | 2.2.3. | 2.2.3. |
| *Secret Data* | 2.2.7. | 2.2.7. | 2.2.7. |
| *Signature Data* | - | - | 2.1.12. |
| *Split Key* | 2.2.5. | 2.2.5. | 2.2.5. |
| *Symmetric Key* | 2.2.2. | 2.2.2. | 2.2.2. |
| *Template* | 2.2.6. | 2.2.6. | 2.2.6. |
| *Template-Attribute Structures* | 2.1.8. | 2.1.8. | 2.1.8. |
| *Transparent DH Private Key* | 2.1.7.6. | 2.1.7.6. | 2.1.7.6. |
| *Transparent DH Public Key* | 2.1.7.7. | 2.1.7.7. | 2.1.7.7. |
| *Transparent DSA Private Key* | 2.1.7.2. | 2.1.7.2. | 2.1.7.2. |
| *Transparent DSA Public Key* | 2.1.7.3. | 2.1.7.3. | 2.1.7.3. |
| *Transparent ECDH Private Key* | 2.1.7.10. | 2.1.7.10. | 2.1.7.10. |
| *Transparent ECDH Public Key* | 2.1.7.11. | 2.1.7.11. | 2.1.7.11. |
| *Transparent ECDSA Private Key* | 2.1.7.8. | 2.1.7.8. | 2.1.7.8. |
| *Transparent ECDSA Public Key* | 2.1.7.9. | 2.1.7.9. | 2.1.7.9. |

  

| **Reference Term** | **KMIP 1.0** | **KMIP 1.1** | **KMIP 1.2** |
|---|---|---|---|
| *Transparent ECMQV Private Key* | 2.1.7.12. | 2.1.7.12. | 2.1.7.12. |
| *Transparent ECMQV Public Key* | 2.1.7.13. | 2.1.7.13. | 2.1.7.13. |
| *Transparent Key Structures* | 2.1.7. | 2.1.7. | 2.1.7. |
| *Transparent RSA Private Key* | 2.1.7.4. | 2.1.7.4. | 2.1.7.4. |
| *Transparent RSA Public Key* | 2.1.7.5. | 2.1.7.5. | 2.1.7.5. |
| *Transparent Symmetric Key* | 2.1.7.1. | 2.1.7.1. | 2.1.7.1. |
| **3 Attributes** | | | |
| *Activation Date* | 3.19. | 3.24. | 3.24. |
| *Alternative Name* | - | - | 3.40. |
| *Application Specific Information* | 3.30. | 3.36. | 3.36. |
| *Archive Date* | 3.27. | 3.32. | 3.32. |
| *Attributes* | 3 | 3 | 3 |
| *Certificate Identifier* | 3.9. | 3.13. | 3.13. |
| *Certificate Issuer* | 3.11. | 3.15. | 3.15. |
| *Certificate Length* | - | 3.9. | 3.9. |
| *Certificate Subject* | 3.10. | 3.14. | 3.14. |
| *Certificate Type* | 3.8. | 3.8. | 3.8. |
| *Compromise Date* | 3.25. | 3.30. | 3.30. |
| *Compromise Occurrence Date* | 3.24. | 3.29. | 3.29. |
| *Contact Information* | 3.31. | 3.37. | 3.37. |
| *Cryptographic Algorithm* | 3.4. | 3.4. | 3.4. |
| *Cryptographic Domain Parameters* | 3.7. | 3.7. | 3.7. |
| *Cryptographic Length* | 3.5. | 3.5. | 3.5. |
| *Cryptographic Parameters* | 3.6. | 3.6. | 3.6. |
| *Custom Attribute* | 3.33. | 3.39. | 3.39. |
| *Deactivation Date* | 3.22. | 3.27. | 3.27. |
| *Default Operation Policy* | 3.13.2. | 3.18.2. | 3.18.2. |
| *Default Operation Policy for Certificates and Public Key Objects* | 3.13.2.2. | 3.18.2.2. | 3.18.2.2. |
| *Default Operation Policy for Secret Objects* | 3.13.2.1. | 3.18.2.1. | 3.18.2.1. |
| *Default Operation Policy for Template Objects* | 3.13.2.3. | 3.18.2.3. | 3.18.2.3. |
| *Destroy Date* | 3.23. | 3.28. | 3.28. |
| *Digest* | 3.12. | 3.17. | 3.17. |
| *Digital Signature Algorithm* | - | 3.16. | 3.16. |
| *Fresh* | - | 3.34. | 3.34. |
| *Initial Date* | 3.18. | 3.23. | 3.23. |
| *Key Value Location* | - | - | 3.42. |
| *Key Value Present* | - | - | 3.41. |
| *Last Change Date* | 3.32. | 3.38. | 3.38. |
| *Lease Time* | 3.15. | 3.20. | 3.20. |
| *Link* | 3.29. | 3.35. | 3.35. |

  

| **Reference Term** | **KMIP 1.0** | **KMIP 1.1** | **KMIP 1.2** |
|---|---|---|---|
| *Name* | 3.2. | 3.2. | 3.2. |
| *Object Group* | 3.28. | 3.33. | 3.33. |
| *Object Type* | 3.3. | 3.3. | 3.3. |
| *Operation Policy Name* | 3.13. | 3.18. | 3.18. |
| *Operations outside of operation policy control* | 3.13.1. | 3.18.1. | 3.18.1. |
| *Original Creation Date* | - | - | 3.43. |
| *Process Start Date* | 3.20. | 3.25. | 3.25. |
| *Protect Stop Date* | 3.21. | 3.26. | 3.26. |
| *Revocation Reason* | 3.26. | 3.31. | 3.31. |
| *State* | 3.17. | 3.22. | 3.22. |
| *Unique Identifier* | 3.1. | 3.1. | 3.1. |
| *Usage Limits* | 3.16. | 3.21. | 3.21. |
| *X.509 Certificate Identifier* | - | 3.10. | 3.10. |
| *X.509 Certificate Issuer* | - | 3.12. | 3.12. |
| *X.509 Certificate Subject* | - | 3.11. | 3.11. |
| **4 Client-to-Server Operations** | | | |
| *Activate* | 4.18. | 4.19. | 4.19. |
| *Add Attribute* | 4.13. | 4.14. | 4.14. |
| *Archive* | 4.21. | 4.22. | 4.22. |
| *Cancel* | 4.25. | 4.27. | 4.27. |
| *Certify* | 4.6. | 4.7. | 4.7. |
| *Check* | 4.9. | 4.10. | 4.10. |
| *Create* | 4.1. | 4.1. | 4.1. |
| *Create Key Pair* | 4.2. | 4.2. | 4.2. |
| *Create Split Key* | - | - | 4.38. |
| *Decrypt* | - | - | 4.30. |
| *Delete Attribute* | 4.15. | 4.16. | 4.16. |
| *Derive Key* | 4.5. | 4.6. | 4.6. |
| *Destroy* | 4.20. | 4.21. | 4.21. |
| *Discover Versions* | - | 4.26. | 4.26. |
| *Encrypt* | - | - | 4.29. |
| *Get* | 4.10. | 4.11. | 4.11. |
| *Get Attribute List* | 4.12. | 4.13. | 4.13. |
| *Get Attributes* | 4.11. | 4.12. | 4.12. |
| *Get Usage Allocation* | 4.17. | 4.18. | 4.18. |
| *Hash* | - | - | 4.37. |
| *Join Split Key* | - | - | 4.39. |
| *Locate* | 4.8. | 4.9. | 4.9. |
| *MAC* | - | - | 4.33. |
| *MAC Verify* | - | - | 4.34. |
| *Modify Attribute* | 4.14. | 4.15. | 4.15. |

  

| Reference Term | KMIP 1.0 | KMIP 1.1 | KMIP 1.2 |
|---|---|---|---|
| *Obtain Lease* | 4.16. | 4.17. | 4.17. |
| *Poll* | 4.26. | 4.28. | 4.28. |
| *Query* | 4.24. | 4.25. | 4.25. |
| *Re-certify* | 4.7. | 4.8. | 4.8. |
| *Recover* | 4.22. | 4.23. | 4.23. |
| *Register* | 4.3. | 4.3. | 4.3. |
| *Re-key* | 4.4. | 4.4. | 4.4. |
| *Re-key Key Pair* | - | 4.5. | 4.5. |
| *Revoke* | 4.19. | 4.20. | 4.20. |
| *RNG Retrieve* | - | - | 4.35. |
| *RNG Seed* | - | - | 4.36. |
| *Sign* | - | - | 4.31. |
| *Signature Verify* | - | - | 4.32. |
| *Validate* | 4.23. | 4.24. | 4.24. |
| **5 Server-to-Client Operations** | | | |
| *Notify* | 5.1. | 5.1. | 5.1. |
| *Put* | 5.2. | 5.2. | 5.2. |
| **6 Message Contents** | | | |
| *Asynchronous Correlation Value* | 6.8. | 6.8. | 6.8. |
| *Asynchronous Indicator* | 6.7. | 6.7. | 6.7. |
| *Attestation Capable Indicator* | - | - | 6.17. |
| *Batch Count* | 6.14. | 6.14. | 6.14. |
| *Batch Error Continuation Option* | 6.13. | 6.13. | 6.13. |
| *Batch Item* | 6.15. | 6.15. | 6.15. |
| *Batch Order Option* | 6.12. | 6.12. | 6.12. |
| *Maximum Response Size* | 6.3. | 6.3. | 6.3. |
| *Message Extension* | 6.16. | 6.16. | 6.16. |
| *Operation* | 6.2. | 6.2. | 6.2. |
| *Protocol Version* | 6.1. | 6.1. | 6.1. |
| *Result Message* | 6.11. | 6.11. | 6.11. |
| *Result Reason* | 6.10. | 6.10. | 6.10. |
| *Result Status* | 6.9. | 6.9. | 6.9. |
| *Time Stamp* | 6.5. | 6.5. | 6.5. |
| *Unique Batch Item ID* | 6.4. | 6.4. | 6.4. |
| **7 Message Format** | | | |
| *Message Structure* | 7.1. | 7.1. | 7.1. |
| *Operations* | 7.2. | 7.2. | 7.2. |
| **8 Authentication** | | | |
| *Authentication* | 8 | 8 | 8 |

  

| **Reference Term** | **KMIP 1.0** | **KMIP 1.1** | **KMIP 1.2** |
|---|---|---|---|
| **9 Message Encoding** | | | |
| *Alternative Name Type Enumeration* | - | - | 9.1.3.2.34. |
| *Attestation Type Enumeration* | - | - | 9.1.3.2.36. |
| *Batch Error Continuation Option Enumeration* | 9.1.3.2.29. | 9.1.3.2.30. | 9.1.3.2.30. |
| *Bit Masks* | 9.1.3.3. | 9.1.3.3. | 9.1.3.3. |
| *Block Cipher Mode Enumeration* | 9.1.3.2.13. | 9.1.3.2.14. | 9.1.3.2.14. |
| *Cancellation Result Enumeration* | 9.1.3.2.24. | 9.1.3.2.25. | 9.1.3.2.25. |
| *Certificate Request Type Enumeration* | 9.1.3.2.21. | 9.1.3.2.22. | 9.1.3.2.22. |
| *Certificate Type Enumeration* | 9.1.3.2.6. | 9.1.3.2.6. | 9.1.3.2.6. |
| *Credential Type Enumeration* | 9.1.3.2.1. | 9.1.3.2.1. | 9.1.3.2.1. |
| *Cryptographic Algorithm Enumeration* | 9.1.3.2.12. | 9.1.3.2.13. | 9.1.3.2.13. |
| *Cryptographic Usage Mask* | 9.1.3.3.1. | 9.1.3.3.1. | 9.1.3.3.1. |
| *Defined Values* | 9.1.3. | 9.1.3. | 9.1.3. |
| *Derivation Method Enumeration* | 9.1.3.2.20. | 9.1.3.2.21. | 9.1.3.2.21. |
| *Digital Signature Algorithm Enumeration* | - | 9.1.3.2.7. | 9.1.3.2.7. |
| *Encoding Option Enumeration* | - | 9.1.3.2.32. | 9.1.3.2.32. |
| *Enumerations* | 9.1.3.2. | 9.1.3.2. | 9.1.3.2. |
| *Examples* | 9.1.2. | 9.1.2. | 9.1.2. |
| *Hashing Algorithm Enumeration* | 9.1.3.2.15. | 9.1.3.2.16. | 9.1.3.2.16. |
| *Item Length* | 9.1.1.3. | 9.1.1.3. | 9.1.1.3. |
| *Item Tag* | 9.1.1.1. | 9.1.1.1. | 9.1.1.1. |
| *Item Type* | 9.1.1.2. | 9.1.1.2. | 9.1.1.2. |
| *Item Value* | 9.1.1.4. | 9.1.1.4. | 9.1.1.4. |
| *Key Compression Type Enumeration* | 9.1.3.2.2. | 9.1.3.2.2. | 9.1.3.2.2. |
| *Key Format Type Enumeration* | 9.1.3.2.3. | 9.1.3.2.3. | 9.1.3.2.3. |
| *Key Role Type Enumeration* | 9.1.3.2.16. | 9.1.3.2.17. | 9.1.3.2.17. |
| *Key Value Location Type Enumeration* | - | - | 9.1.3.2.35. |
| *Link Type Enumeration* | 9.1.3.2.19. | 9.1.3.2.20. | 9.1.3.2.20. |
| *Name Type Enumeration* | 9.1.3.2.10. | 9.1.3.2.11. | 9.1.3.2.11. |
| *Object Group Member Enumeration* | - | 9.1.3.2.33. | 9.1.3.2.33. |
| *Object Type Enumeration* | 9.1.3.2.11. | 9.1.3.2.12. | 9.1.3.2.12. |
| *Opaque Data Type Enumeration* | 9.1.3.2.9. | 9.1.3.2.10. | 9.1.3.2.10. |
| *Operation Enumeration* | 9.1.3.2.26. | 9.1.3.2.27. | 9.1.3.2.27. |
| *Padding Method Enumeration* | 9.1.3.2.14. | 9.1.3.2.15. | 9.1.3.2.15. |
| *Put Function Enumeration* | 9.1.3.2.25. | 9.1.3.2.26. | 9.1.3.2.26. |
| *Query Function Enumeration* | 9.1.3.2.23. | 9.1.3.2.24. | 9.1.3.2.24. |
| *Recommended Curve Enumeration for ECDSA, ECDH, and ECMQV* | 9.1.3.2.5. | 9.1.3.2.5. | 9.1.3.2.5. |
| *Result Reason Enumeration* | 9.1.3.2.28. | 9.1.3.2.29. | 9.1.3.2.29. |
| *Result Status Enumeration* | 9.1.3.2.27. | 9.1.3.2.28. | 9.1.3.2.28. |
| *Revocation Reason Code Enumeration* | 9.1.3.2.18. | 9.1.3.2.19. | 9.1.3.2.19. |

  

| Reference Term | KMIP 1.0 | KMIP 1.1 | KMIP 1.2 |
|---|---|---|---|
| *Secret Data Type Enumeration* | 9.1.3.2.8. | 9.1.3.2.9. | 9.1.3.2.9. |
| *Split Key Method Enumeration* | 9.1.3.2.7. | 9.1.3.2.8. | 9.1.3.2.8. |
| *State Enumeration* | 9.1.3.2.17. | 9.1.3.2.18. | 9.1.3.2.18. |
| *Storage Status Mask* | 9.1.3.3.2. | 9.1.3.3.2. | 9.1.3.3.2. |
| *Tags* | 9.1.3.1. | 9.1.3.1. | 9.1.3.1. |
| *TTLV Encoding* | 9.1. | 9.1. | 9.1. |
| *TTLV Encoding Fields* | 9.1.1. | 9.1.1. | 9.1.1. |
| *Usage Limits Unit Enumeration* | 9.1.3.2.30. | 9.1.3.2.31. | 9.1.3.2.31. |
| *Validity Indicator Enumeration* | 9.1.3.2.22. | 9.1.3.2.23. | 9.1.3.2.23. |
| *Wrapping Method Enumeration* | 9.1.3.2.4. | 9.1.3.2.4. | 9.1.3.2.4. |
| *XML Encoding* | 9.2. | - | - |
| **10 Transport** | | | |
| *Transport* | 10 | 10 | 10 |
| **12 KMIP Server and Client Implementation Conformance** | | | |
| *Conformance clauses for a KMIP Server* | 12.1. | - | - |
| *KMIP Client Implementation Conformance* | - | 12.2. | 12.2. |
| *KMIP Server Implementation Conformance* | - | 12.1. | 12.1. |

  

# Appendix E. Revision History

| Revision | Date | Editor | Changes Made |
|---|---|---|---|
| V1.2-wd01-01 | 3/18/13 | Indra Fitzgerald | Conversion of UG into current OASIS template |
| V1.2-wd01-02 | 5/9/13 | Judy Furlong | Restructuring of UG<br>• Split section 3 into two section (Using vs. Applying KMIP functionality)<br>  ◦ Section 3.18 Locate Queries now 4.1<br>  ◦ Section 3.21 Using Wrapped Keys now 4.2<br>  ◦ Section 3.30 Interoperable Key Naming for Tape now section 4.3<br>  ◦ Section 3.37 Vendor Extensions – the intro remains in Section 3 (now section 3.34) and the subsections (3.37.1 and 3.37.2) moved to Section 4 (4.4 and 4.5 respectively<br>• Added a deprecation section<br>• Removed the deferred item section (going to Use Case document).<br>Other editorial changes. |
| V1.2-wd01-03 | 5/16/13 | Judy Furlong | Incorporation of the following balloted proposals:<br>• Metadata-only Object<br>• Deprecation Rule<br>• PGP and Alternative Name<br>• Attested Operations<br>• Split Key<br>Incorporation of other UG related content:<br>• Compromised State of Linked Objects<br>• Client Registration<br>• PGP Cert and Cert Type Deprecation |
| V1.2-wd02 | 5/30/13 | Judy Furlong | Incorporation of the UG text for the following balloted proposals:<br>• Templates |

  

| | | | |
|---|---|---|---|
| | | | • Cryptographic Services<br>Other UG related content changes:<br>• Application Specification Information<br>• Removed Compromised State of Linked Objects section until wording can be agreed upon<br>• Incorporated 1.1 Errata for the Usage Guide |
| V1.2-wd03 | 6/27/13 | Judy Furlong | Other UG related content changes:<br>• Updated section 4.3 Tape Key Name Space to bring in-line with profile<br>• Readded Compromised Objects section<br>• Updated participant list<br>• Other editorial changes |
| V1.2-wd04 | 7/11/13 | Judy Furlong | • Incorporated review comments from TC members<br>• Added ECC Algorithm Mapping information to close out incorporation of all KMIP 1.2 balloted proposals.<br>• Removed Acronym list in appendix (a single list will be included in the KMIP Specification)<br>• Other editorial and format changes. |
| V1.2-wd05 | 8/19/13 | Judy Furlong | • Incorporated review comments from TC members<br>• Added new version of ECC Algorithm Mapping table.<br>• Edits to 3.5 to align with latest KMIP Spec wording.<br>• Other editorial changes |
| V1.2-wd06 | 8/22/13 | Judy Furlong | • Updated References<br>• Re-added Acronym List<br>• Other editorial and format changes |

  

| | | | |
|---|---|---|---|
| V1.2-cnd01 | 9/13/13 | Judy Furlong | Converted to Committee Note Draft<br>Incorporated applicable updated references<br>Incorporated updated Participants List<br>Fixed Cross-references |
| V1.2-cnd0x | 6/13/14 | Judy Furlong | Incorporated comments from Initial Public Review<br>Removed references to KMIP 1.3 Use Case document. |
| [Rev number] | [Rev Date] | [Modified By] | [Summary of Changes] |

  