Key Management Interoperability Protocol Usage Guide Version 1.1

Committee Note 01

27 July 2012

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

Specification URIs

This version:

http://docs.oasis-open.org/kmip/ug/v1.1/cn01/kmip-ug-v1.1-cn01.doc (Authoritative)
http://docs.oasis-open.org/kmip/ug/v1.1/cn01/kmip-ug-v1.1-cn01.html
http://docs.oasis-open.org/kmip/ug/v1.1/cn01/kmip-ug-v1.1-cn01.pdf

Previous version:

http://www.oasis-open.org/committees/download.php/44883/kmip-ug-v1.1-cnprd01.zip

Latest version:

http://docs.oasis-open.org/kmip/ug/v1.1/kmip-ug-v1.1.doc (Authoritative)
http://docs.oasis-open.org/kmip/ug/v1.1/kmip-ug-v1.1.html
http://docs.oasis-open.org/kmip/ug/v1.1/kmip-ug-v1.1.pdf

Technical Committee:

OASIS Key Management Interoperability Protocol (KMIP) TC

Chairs:

Robert Griffin (robert.griffin@rsa.com), EMC Corporation
Subhash Sankuratripati (Subhash.Sankuratripati@netapp.com), NetApp

Editors:

Indra Fitzgerald (indra.fitzgerald@hp.com), HP
Robert Griffin (robert.griffin@rsa.com), EMC Corporation

Related work:

This document replaces or supersedes:

- *Key Management Interoperability Protocol Usage Guide Version 1.0*. OASIS Committee Specification 01. 15 June 2010. http://docs.oasis-open.org/kmip/ug/v1.0/cs01/kmip-ug-1.0-cs-01.html.

This document is related to:

- *Key Management Interoperability Protocol Specification Version 1.1*. Latest version. http://docs.oasis-open.org/kmip/spec/v1.1/kmip-spec-v1.1.html

- *Key Management Interoperability Protocol Profiles Version 1.1*. Latest version. http://docs.oasis-open.org/kmip/profiles/v1.1/kmip-profiles-v1.1.html
- *Key Management Interoperability Protocol Test Cases Version 1.1*. Latest version. http://docs.oasis-open.org/kmip/testcases/v1.1/kmip-testcases-v1.1.html

Abstract:

This document is intended to complement the Key Management Interoperability Protocol Specification by providing guidance on how to implement the Key Management Interoperability Protocol (KMIP) most effectively to ensure interoperability.

KMIP V1.1 enhances the KMIP V1.0 standard (established in October 2010) by

1) defining new functionality in the protocol to improve interoperability, such as a Discover Versions operation and a Group object;
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.

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]

Key Management Interoperability Protocol Usage Guide Version 1.1. 27 July 2012. OASIS Committee Note 01.
http://docs.oasis-open.org/kmip/ug/v1.1/cn01/kmip-ug-v1.1-cn01.html.

Table of Contents

1 Introduction

This Key Management Interoperability Protocol Usage Guide Version 1.1 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. 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.
- Functionality considered for inclusion in KMIP V1.1, but deferred to subsequent versions of the standard.

A selected set of conformance profiles and authentication suites are defined in the KMIP Profiles specification **[KMIP-Prof].**

[KMIP-Prof]Further assistance for implementing KMIP is provided by the KMIP Test Cases for Proof of Concept Testing 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 Terminology

1.2 For a list of terminologies refer to **[KMIP-Spec].** Normative References

[FIPS186-3]

Digital Signature Standard (DSS), FIPS PUB 186-3, June 2009, http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

[FIPS197]

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

[FIPS198-1]

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

[IEEE1003-1]

IEEE Std 1003.1, *Standard for information technology - portable operating system interface (POSIX). Shell and utilities*, 2004.

[ISO16609]

ISO, *Banking -- Requirements for message authentication using symmetric techniques*, ISO 16609, 1991.

[ISO9797-1]

ISO/IEC, *Information technology -- Security techniques -- Message Authentication Codes (MACs) -- Part 1: Mechanisms using a block cipher*, ISO/IEC 9797-1, 1999.

[KMIP-Spec]

*Key Management Interoperability Protocol Specification Version 1.1.*Working Draft 07. 27 April 2012. http://www.oasis-open.org/apps/org/workgroup/kmip/download.php/45731/kmip-spec-v1.1-wd06.doc

[KMIP-Prof]

Key Management Interoperability Protocol Profiles Version 1.1. Working Draft 11. 26 April 2012. http://www.oasis-open.org/apps/org/workgroup/kmip/download.php/45854/kmip-profiles-v1.1-wd11.docx

[PKCS#1]

RSA Laboratories, *PKCS #1 v2.1: RSA Cryptography Standard*, June 14, 2002, http://www.rsa.com/rsalabs/node.asp?id=2125

[PKCS#5]

RSA Laboratories, *PKCS #5 v2.1: Password-Based Cryptography Standard*, October 5, 2006, http://www.rsa.com/rsalabs/node.asp?id=2127

[PKCS#7]

RSA Laboratories, *PKCS#7 v1.5: Cryptographic Message Syntax Standard. November 1, 1993,* http://www.rsa.com/rsalabs/node.asp?id=2129

[PKCS#8]

RSA Laboratories, PKCS#8 v1.2: Private-Key Information Syntax Standard, November 1, 1993, http://www.rsa.com/rsalabs/node.asp?id=2130

[PKCS#10]

RSA Laboratories, *PKCS #10 v1.7: Certification Request Syntax Standard*, May 26, 2000, http://www.rsa.com/rsalabs/node.asp?id=2132

[RFC1319]

B. Kaliski, *The MD2 Message-Digest Algorithm*, IETF RFC 1319, Apr 1992, http://www.ietf.org/rfc/rfc1319.txt

[RFC1320]

R. Rivest, *The MD4 Message-Digest Algorithm*, IETF RFC 1320, Apr 1992, http://www.ietf.org/rfc/rfc1320.txt

[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

[RFC1424]

B. Kaliski, *Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services*, IETF RFC 1424, February 1993, http://www.ietf.org/rfc/rfc1424.txt

[RFC2104]

H. Krawczyk, M. Bellare, R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, IETF RFC 2104, Feb 1997, http://www.ietf.org/rfc/rfc2104.txt

[RFC2119]

S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.

[RFC2253]

M. Wahl, S. Kille, T. Howes, Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names, IETF RFC 2253, Dec 1997, http://www.ietf.org/rfc/rfc2253.txt

[RFC2898]

B. Kaliski, *PKCS #5: Password-Based Cryptography Specification Version 2.0*, IETF RFC 2898, Sep 2000, http://www.ietf.org/rfc/rfc2898.txt

[RFC3394]

J. Schaad, R. Housley, *Advanced Encryption Standard (AES) Key Wrap Algorithm*, IETF RFC 3394, Sep 2002, http://www.ietf.org/rfc/rfc3394.txt

[RFC3447]

J. Jonsson, B. Kaliski, *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*, IETF RFC 3447 Feb 2003, http://www.ietf.org/rfc/rfc3447.txt

[RFC3629]

F. Yergeau, *UTF-8, a transformation format of ISO 10646*, IETF RFC 3629, Nov 2003, http://www.ietf.org/rfc/rfc3629.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, *RFC2510: Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)*, September 2005, http://www.ietf.org/rfc/rfc4210.txt

[RFC4211]

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

[RFC4868]

S. Kelly, S. Frankel, *Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec*, IETF RFC 4868, May 2007, http://www.ietf.org/rfc/rfc4868.txt

[RFC4880]

J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer, *OpenPGP Message Format*, IETF RFC 4880, Nov 2007, http://www.ietf.org/rfc/rfc4880.txt

[RFC4949]

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

[RFC5272]

J. Schaad and M. Meyers, *RFC5272: Certificate Management over CMS (CMC)*, June 2008, http://www.ietf.org/rfc/rfc5272.txt

[RFC5280]

D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and W. Polk, *RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, May 2008, http://www.ietf.org/rfc/rfc5280.txt

[RFC5649]

R. Housley, *Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm*, IETF RFC 5649, Aug 2009, http://www.ietf.org/rfc/rfc5649.txt

[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

[SP800-38B]

M. Dworkin, *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*, NIST Special Publication 800-38B, May 2005, http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf

[SP800-38C]

M. Dworkin, *Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality*, NIST Special Publication 800-38C, May 2004, http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.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

[SP800-38E]

M. Dworkin, *Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Block-Oriented Storage Devices*, NIST Special Publication 800-38E, Jan 2010, http://csrc.nist.gov/publications/nistpubs/800-38E/nist-sp-800-38E.pdf

[SP800-56A]

E. Barker, D. Johnson, and M. Smid, *Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised)*, NIST

Special Publication 800-56A, March 2007, http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.pdf

[SP800-56B]

E. Barker, L. Chen, A. Regenscheid, M. Smid, *Recommendation for Pair-Wise Key Establishment Schemes Using Integer Factorization Cryptography*, NIST Special Publication 800-56B, August 2009, http://csrc.nist.gov/publications/nistpubs/800-56B/sp800-56B.pdf

[SP800-57-1]

E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, *Recommendations for Key Management - Part 1: General (Revised),* NIST Special Publication 800-57 part 1, March 2007, http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf

[SP800-67]

W. Barker, *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher*, NIST Special Publication 800-67, Version 1.1, Revised 19 May 2008, http://csrc.nist.gov/publications/nistpubs/800-67/SP800-67.pdf

[SP800-108]

L. Chen, *Recommendation for Key Derivation Using Pseudorandom Functions (Revised)*, NIST Special Publication 800-108, October 2009, http://csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf

[X.509]

International Telecommunication Union (ITU)–T, X.509: Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks, August 2005, http://www.itu.int/rec/T-REC-X.509-200508-I/en

[X9.24-1]

ANSI, *X9.24: Retail Financial Services Symmetric Key Management - Part 1: Using Symmetric Techniques*, 2004.

[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.

[X9-57]

ANSI, *X9-57: Public Key Cryptography for the Financial Services Industry: Certificate Management*, 1997.

[X9.62]

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

[X9-63]

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

[X9-102]

ANSI, *X9-102: Symmetric Key Cryptography for the Financial Services Industry - Wrapping of Keys and Associated Data*, 2008.

[X9 TR-31]

ANSI, *X9 TR-31: Interoperable Secure Key Exchange Key Block Specification for Symmetric Algorithms*, 2005.

1.3 Non-normative References

[KMIP-TC]

Key Management Interoperability Protocol Test Cases Version 1.1. Committee Note Draft.1 December 2011. http://docs.oasis-open.org/kmip/usecases/v1.1/kmip-usecases-v1.1-cnd01.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],**

[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.

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 and their components
- Digital Certificates
- Derived Keys
- 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 Messages

The protocol allows two modes of operation. Synchronous (mandatory) operations are those in which a client sends a request and waits for a response from the server. Polled 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 may choose not to support the Polled Asynchronous feature of the protocol.

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. An option is provided to indicate whether to continue processing requests after an earlier request in the batch fails or to stop processing the remaining requests in the batch. Note that there is no option to treat an entire batch as atomic, that is, if a request in the batch fails, then preceding requests in the batch are not undone or rolled back (see Section 3.15). A special ID Placeholder (see Section 3.19) 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 NIST SP 800-57 key state definitions **[SP800-57-1]**, 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 Usage Guidelines

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

[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. Alternatively, 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:

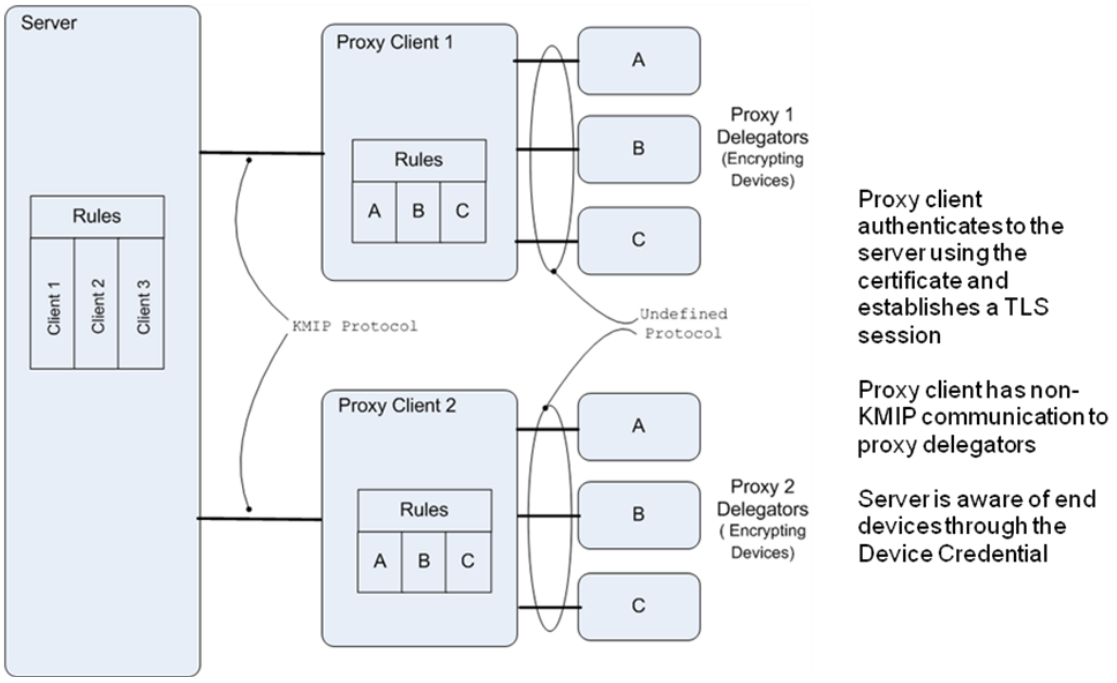

FIGURE 1: AGGREGATOR CLIENT EXAMPLE

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 fibre 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]**

[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. Attribute values other than “Last Change Date” should not be included in a Notify to minimize risk of exposure of attribute information.
- 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 may begin to be used for applying cryptographic protection to data
- Process Start Date: The date and time when a managed symmetric key object may begin to be used for processing cryptographically protected data
- Protect Stop Date: The date and time when a managed symmetric key object may no longer be used for applying cryptographic protection to data
- Deactivation Date: The date and time when the managed cryptographic object may no longer be used for any purpose, except for decryption, signature verification, or unwrapping, but only under extraordinary circumstances and when special permission is granted
- 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 is 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 set any of the following attributes for a managed object:

- Cryptographic Algorithm
- Cryptographic Length
- Cryptographic Domain Parameters
- Cryptographic Parameters
- Operation Policy Name
- Cryptographic Usage Mask
- Certificate Length
- Digital Signature Algorithm
- Usage Limits
- Activation Date
- Process Start Date
- Protect Stop Date
- Deactivation Date
- Object Group
- Application Specific Information
- Contact Information
- Custom Attribute

In addition to these attributes, the template has attributes that are applicable to the template itself. These include the attributes (Unique Identifier, Initial Date, Last Change Date, and Archive

Date) set implicitly after successfully completing a certain operation and attributes set by the client (Object Type and Name) in the Register request. When registering a template, the Name attribute for the template should be set. It is used to specify and identify the template in the Template-Attribute structure when attributes for a managed object are set.

The Template-Attribute structure allows for multiple template names and individual attributes 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.

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. Clients are not able to create a template with the Create operation; instead templates are created using the Register operation. When the template is the subject of the operation, the Unique ID is used to identify the template. The template name is only used to identify the template 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 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:
  - Name: Template1
  - Cryptographic Algorithm: AES
  - Cryptographic Length: 256
  - Cryptographic Usage Mask: Encrypt and Decrypt
  - Operation Policy Name: 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:

    Name: AESkey

    Custom Attribute: x-ID74592

The Template-Attribute structure specifies both a template name and additional attributes. The Name attribute is not an attribute that may be set by a template. The Name attribute set for the template applies to the template itself (e.g., Template1 is the Name attribute of the Template object). The Name attribute for the symmetric key is therefore specified separately under Attribute. It is possible to specify the Custom Attribute inside the template when the template is registered; however, this particular example sets this attribute separately.

3.7 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

780 those cases in which it is possible to assure the client, through mechanisms outside of KMIP, of
781 the server effectively supporting atomicity for batch operations.

782 3.16 Maximum Message Size

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

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

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

793 3.17 Using Offset in Re-key and Re-certify Operations

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

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

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

805 Note that in re-key operations if activation date, process start date, protect stop date and
806 deactivation date are obtained from the existing key, and the initial date is obtained from the
807 current time, then the deactivation/activation date/process start date/protect stop date is
808 smaller or less than initial date. KMIP allows back-dating of these values to prevent this
809 contradiction (see KMIP-Spec 3.22 and KMIP-UG 3.5 and 3.29).

810

811 3.18 Locate Queries

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

813 • Exact match of a transition to a given state. Locate the key(s) with a transition to a certain
814 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.

3.19 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)
Register	-	ID of newly registered Object
Derive Key	- (multiple Unique Identifiers may be specified in the request)	ID of new Symmetric Key
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

882 **TABLE 1: ID PLACEHOLDER PRIOR TO AND RESULTING FROM A KMIP OPERATION**

883 3.20 Key Block

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

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

896 • Unique Identifier – uniquely identifies the key
897 • Cryptographic Algorithm (e.g., AES, 3DES, RSA) – this attribute is either specified inside
898 the Key Block structure or the Key Value structure
899 • Cryptographic Length (e.g., 128, 256, 2048) – this attribute is either specified inside the
900 Key Block structure or the Key Value structure
901 • Cryptographic Usage Mask– identifies the cryptographic usage of the key (e.g., Encrypt,
902 Wrap Key, Export)
903 • Cryptographic Parameters – provides additional parameters for determining how the key
904 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 financial 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

3.21 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.

3.21.1 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.

3.21.2 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 3.21.1 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.

3.21.3 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.

3.21.4 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.

3.21.5 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.

3.21.6 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.

3.22 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]**. 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.

3.23 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 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.

3.24 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

3.24.1 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:

Common Template-Attribute

- RSACom Template
  - Cryptographic Algorithm: RSA
  - Cryptographic Length: 2048
  - Cryptographic Parameters: Padding Method OAEP
  - Custom Attribute: x-Serial 1234
  - Object Group: Key encryption group 1
- Attribute
  - Cryptographic Length: 4096
  - Cryptographic Parameters: Padding Method PKCS1 v1.5
  - Custom Attribute: x-ID 56789

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

Private Key Template-Attribute

- RSAPriv Template
  - Object Group: Key encryption group 2
- Attribute
  - Cryptographic Usage Mask: Unwrap Key
  - Name: PrivateKey1

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

Public Key Template-Attribute

- Attribute
  - Cryptographic Usage Mask: Wrap Key
  - Name: PublicKey1

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

Private Key

- Cryptographic Algorithm: RSA
- Cryptographic Length: 4096
- Cryptographic Parameters: OAEP
- Cryptographic Parameters: PKCS1 v1.5
- Cryptographic Usage Mask: Unwrap Key
- Custom Attribute: x-Serial 1234
- Custom Attribute: x-ID 56789
- Object Group: Key encryption group 1
- Object Group: Key encryption group 2
- Name: PrivateKey1

Public Key

- Cryptographic Algorithm: RSA
- Cryptographic Length: 4096
- Cryptographic Parameters: OAEP
- Cryptographic Parameters: PKCS1 v1.5
- Cryptographic Usage Mask: Wrap Key
- Custom Attribute: x-Serial 1234
- Custom Attribute: x-ID 56789
- Object Group: Key encryption group 1
- Name: PublicKey1

3.25 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
2. 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

3.26 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

3.27 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 generate 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 generate and verify, 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 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 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 TRANSLATE DECRYPT TRANSLATE WRAP 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. *TRANSLATE ENCRYPT/DECRYPT* is used for data encipherment. *TRANSLATE WRAP/UNWRAP* is used for key wrapping.

1314 **TABLE 2: CRYPTOGRAPHIC USAGE MASKS PAIRS**

1315 In order to support asymmetric concepts using symmetric keys in a KMIP system, the server
1316 implementation needs to be able to differentiate between clients for generate operations and
1317 clients for verify operations. As indicated by Section 3 ("Attributes") of

1318 [KMIP-Spec] there is a single key object in the system to which all relevant clients refer, but
1319 when a client requests that key, the server is able to choose which attributes (permissions) to
1320 send with it, based on the identity and configured access rights of that specific client. There is,
1321 thus, no need to maintain and synchronize distinct copies of the symmetric key – just a need to
1322 define access policy for each client or group of clients.

1323 The internal implementation of this feature at the server end is a matter of choice for the
1324 vendor: storing multiple key blocks with all necessary combinations of attributes or generating
1325 key blocks dynamically are both acceptable approaches.

1326 3.28 Application Specific Information

1327 The Application Specific Information attribute is used to store data which is specific to the
1328 application(s) using the object. Some examples of Application Namespace and Application Data
1329 pairs are given below.

1330 • SMIME, 'someuser@company.com'
1331 • TLS, 'some.domain.name'
1332 • Volume Identification, '123343434'
1333 • File Name, 'secret.doc'
1334 • Client Generated Key ID, '450994003'

The following Application Namespaces are recommended:

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

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 set to null, as shown in the example below, and the Application Namespace is recognized by the server, the server does not generate the Application Data for the client. 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});
```

3.29 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].**

.

3.30 Interoperable Key Naming for Tape

This section describes methods for creating and storing key identifiers that are interoperable across multi-vendor KMIP clients.

3.30.1 Native Tape Encryption by a KMIP Client

This method is primarily intended to promote interoperable key naming between tape library products which already support non-KMIP key managers, where KMIP support is being added.

When those existing library products become KMIP clients, a common method for naming and storing keys may be used to support moving tape cartridges between the libraries, and successfully retrieving keys, assuming that the clients have appropriate access privileges. The library clients may be from multiple vendors, and may be served by a KMIP key manager from a different vendor.

3.30.1.1 Method Overview

- The method uses the KMIP Application Specific Information (ASI) attribute's Application Data field to store the key name. The ASI Application Namespace is used to identify the namespace (such as LIBRARY-LTO4 or LIBRARY-LTO5).
- The method also uses the tape format's Key Associated Data (KAD) fields to store the key name. Tape formats may provide both authenticated and unauthenticated storage for the KAD data. This method ensures optimum utilization of the authenticated KAD data when the tape format supports authentication.
- The method supports both client-generated and server-generated key names.
- The method, in many cases, is backward-compatible if tapes are returned to a non-KMIP key manager environment.
- Key names stored in the KMIP server's ASI attribute are always ASCIIformat. Key names stored on the KMIP client's KAD fields are always numeric format, due to space limitations of the tape format. The method basically consists of implementing a specific algorithm for converting between text and numeric formats.

- The algorithm used by this conversion is reversible.

3.30.1.2 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's combined KAD fields (both authenticated and unauthenticated).
- N(a), N(u). The number of bytes in the tape format's authenticated, and unauthenticated KAD fields, respectively.

3.30.1.3 Algorithm 1. Numeric to text direction (tape format's KAD to KMIP ASI)

Description: All information contained in the tape format's KAD fields is converted to a null-terminated ASCII string consisting of hexadecimal numeric character pairs. First, the unauthenticated KAD data is converted to text. Then, the authenticated KAD data is converted and appended to the end of the string. The string is then null-terminated.

Implementation Example:

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 null-terminated string with a maximum length of 2*N(k)+1 bytes.
3. Define the standard POSIX (also known as C) locale. Each character in the string is a single-byte US-ASCII character.
4. Copy the tape format's KAD data, from the unauthenticated KAD field first, to the input buffer. Effectively, the first byte (byte 0) of the input buffer is the first byte of unauthenticated KAD. Bytes from the authenticated KAD are concatenated, after the unauthenticated bytes.
5. For each byte in the input buffer, convert to US-ASCII as follows:
   a. 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.
   b. After all byte values have been converted, null terminate the output buffer.
6. When storing the string to the KMIP server, use the object's ASI attribute's Application Data field. Store the namespace (such as LIBRARY-LTO4) in the ASI attribute's Application Namespace field.

3.30.1.4 Algorithm 2. Text to numeric direction (KMIP ASI to tape format's KAD)

Description: Hexadecimal numeric character pairs in the null-terminated ASCII string are converted to single byte numeric values, and stored in the tape format's KAD fields. The authenticated KAD field is populated first, from a sub-string consisting of the last 2*N(a) characters in the full string. Any remaining characters in the string are converted and stored to the unauthenticated KAD field. The null termination byte is not converted.

Implementation Example:

1. Obtain the key's name from the KMIP server's ASI attribute for that object. Copy the null terminated string to an input buffer of size 2*N(k) + 1 bytes. For LTO4, an 89 character string, including null termination, is sufficient to represent any key name stored directly in the KAD fields. For LTO5, a 195 character string, including null termination, 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 2*N(a) characters of the full string, not including the null termination byte.
5. When the authenticated KAD is filled, next populate the unauthenticated KAD buffer, by converting the remaining hexadecimal character pairs in the string.

3.30.1.5 Example Output

The following are examples illustrating some results of this method. In the following examples, the sizes of the KAD for LTO4 are used. Different tape formats may utilize different KAD sizes.

Example 1. Full combined KAD for LTO4

This LTO4 tape's combined KAD contains the following data (represented in hexadecimal). For LTO4, the unauthenticated KAD contains 32 bytes, and the authenticated KAD contains 12 bytes.

Example 1a. Hexadecimal numeric data from a tape's KAD.

Shaded data is authenticated by the tape drive.

02 04 17 11 39 43 42 36 30 41 33 34 39 31 44 33

41 41 43 36 32 42 07 F6 54 54 32 36 30 38 4C 34

30 30 30 39 30 35 32 38 30 34 31 32

The algorithm converts the numeric KAD data to the following 89 character null-terminated string for storage in the Application Data field of a KMIP object's Application Specific Information attribute. The ASI Application Namespace contains "LIBRARY-LTO4".

Example 1b. Text string from KMIP ASI Application Data.

Shaded characters are derived from authenticated data. The null character is represented as <null>

020417113943423630413334393144334141433632420 7F65454323630384C343030303930353238303431 32<null>

Example 1c. The hexadecimal values of the 89 US-ASCII characters in string 1b, from the KMIP ASI Application Data. Note: these values are always in the range 30h-39h, or in the range 41h-46h, or the 0h null.

30 32 30 34 31 37 31 31 33 39 34 33 34 32 33 36 33 30 34 31 33 33 33 34 33 39 33 31 34 34 33 33 34 31 34 31 34 33 33 36 33 32 34 32 30 37 46 36 35 34 35 34 33 32 33 36 33 30 33 38 34 43 33 34 33 30 33 30 33 30 33 39 33 30 33 35 33 32 33 38 33 30 33 34 33 31 33 32 00

For the reverse transformation, a client would retrieve the string in 1b from the server, derive the numeric values shown in 1a, and store them to the tape format's KAD data. First, the sub-string containing the right-most 24 characters of the full 1b string are used to derive the 12-byte authenticated KAD. The remaining characters are used to derive the 32-byte unauthenticated KAD.

Example 2. Authenticated KAD only, for LTO4

This LTO4 tape's KAD contains the following data (represented in hexadecimal), all 12 bytes obtained from the authenticated KAD field. There is no unauthenticated KAD data.

Example 2a. Hexadecimal numeric data from a tape's KAD.

Shaded data is authenticated.

17 48 33 C6 20 42 10 A7 E8 05 F8 C7

The algorithm converts the numeric KAD data to the following 24 character null-terminated string, for storage in the Application Data field of a KMIP object's Application Specific Information attribute.

Example 2b. Text string from KMIP ASI Application Data.

Shaded characters are derived from authenticated data. The null character is represented as <null>

174833C6204210A7E805F8C7<null>

For the reverse transformation, a client would derive the numeric values in 2a, and store them to the tape format's KAD data. The right-most 24 characters of the string in 2b are used to derive the 12 byte authenticated KAD. In this example, there is no unauthenticated KAD data.

Example 3. Partially filled authenticated KAD originating from a non-KMIP method, for LTO4

This LTO4 tape's KAD contains the following data (represented in hexadecimal). The unauthenticated KAD contains 10 bytes, and the authenticated KAD contains 8 bytes.

Since the authenticated KAD was not filled, but the unauthenticated data was populated, the method creating this key name is potentially not backward-compatible with the KMIP key naming method. See backward-compatibility assessment, below.

Example 3a. Hexadecimal numeric data from a non-KMIP tape's KAD.

Shaded data is authenticated.

02 04 17 11 39 43 42 36 30 41 30 30 30 39 30 35

32 38

The algorithm converts the numeric KAD data to the following 36 character null-terminated string, for storage in the Application Data field of a KMIP object's Application Specific Information attribute.

Example 3b. Text string from KMIP ASI Application Data.

Shaded characters are derived from authenticated data. The null character is represented as <null>

02041711394342363041303030393035323 8<null>

For the reverse transformation, a client would derive the same numeric values shown in 3a, and store them to the tape's KAD. But their storage locations within the KAD now differs (see 3c). The right-most 24 characters from the text string in 3b are used to derive the 12-byte authenticated KAD. The remaining characters are used to fill the 32-byte unauthenticated KAD.

Example 3c. Hexadecimal numeric data from a tape's KAD.

Shaded data is authenticated.

02 04 17 11 39 43 42 36 30 41 30 30 30 39 30 35

32 38

3.30.1.6 Backward-compatibility assessment

Where all the following conditions exist, a non-KMIP solution may encounter compatibility issues during the Read and Appended Write use cases.

1. The tape format supports authenticated KAD, but the non-KMIP solution does not use, or only partially uses, the authenticated KAD field.
2. The non-KMIP solution is sensitive to data position within the combined KAD.
3. The media was written in a KMIP environment, using this method, then moved to the non-KMIP environment.

3.31 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 RFC 5280 with the following exceptions. The *certificateHold* and *removeFromCRL* reason codes have been excluded from [KMIP-Spec], since this version of 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 *priviledgeWithdrawn* 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.

3.32 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 RFCs **[RFC3647]** **[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.

3.33 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

[FIPS197] and Triple-DES
[SP800-67**]**.

3.33.1 AES Key Encoding

[FIPS197] 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

[FIPS197] 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

[FIPS197] 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.

3.33.2 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

3.34 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 186-3 **[FIPS186-3]**, ANSI X9.42 **[X9.42]**, etc) from which the key formats are obtained and the best practice documents (e.g., NIST SP800-57 part 1 **[SP800-57-1]**, NIST SP800-56A **[SP800-56A]**, etc) which recommend that a key pair only be used for one purpose.

3.35 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.

3.36 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.

3.37 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

This section covers only item 2.

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.

3.37.1 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

3.37.2 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.

3.38 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 are 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 is being 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 are being introduced. Certificate attributes for PGP certificates may be introduced in a subsequent (post v1.1) release of KMIP.

1740 Table 3 provides a list of the deprecated certificate-related attributes and fields with their
1741 corresponding tag value.

1742

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

1743

1744 **TABLE 3: DEPRECATED CERTIFICATE RELATED ATTRIBUTES AND FIELDS**

1745

1746

1747 Table 4 provides a mapping of v1.0 to v1.1 certificate attributes and fields.

1748

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

1749

1750 **TABLE 4: MAPPING OF V1.0 TO V1.1 CERTIFICATE RELATED ATTRIBUTES AND FIELDS**

1751 3.39 Certificate Revocation Lists

1752 Any Certificate Revocation List (CRL) checking which may be required for certificate-related
1753 operations such as register and re-key should be performed by the client prior to requesting the
1754 operation from a server.

1755

1756 3.40 Using the "Raw" Key Format Type

1757 As defined in Section 2.1.3 of the KMIP Specification V1.1, the "raw" key format is intended to
1758 be used for "a key that contains only cryptographic key material, encoded as a string of bytes."
1759 As discussed in Section 3.21.6 of the Usage Guide, the "raw" key format supports situations
1760 such as "non-KMIP-aware end-clients are aware how wrapped cryptographic objects (possibly
1761 Raw keys) from the KMIP server should be used without having to rely on the attributes
1762 provided by the Get Attributes operation" and in that regard is similar to the Opaque key format
1763 type. "Raw" key format is intended to be applied to symmetric keys and not asymmetric keys;
1764 therefore, this format is not specified in the asymmetric key profiles included in KMIP V1.1.

1765

1766 3.41 Deprecated Functionality

1767 Use of deprecated functionality, as described in Section 3.39, is discouraged since such
1768 functionality may be dropped in a future release of the KMIP standard.

1769

4 Deferred KMIP Functionality

The KMIP Specification is currently missing items that have been judged candidates for future inclusion in the specification. These items currently include:

- Registration of Clients. This would allow in-band registration and management of clients, which currently may only be registered and/or managed using off-line mechanisms.
- Client-requested specification of additional clients that are allowed to use a key. This requires coordinated identities between the client and server, and as such, is deferred until registration of clients is addressed.
- Registration of Notifications. This would allow clients to specify, using an in-band mechanism, information and events that they wish to be notified of, and what mechanisms should be used for such notifications, possibly including the configuration of pushed cryptographic material. This functionality would assume the Registration of Clients as a prerequisite.
- Key Migration. This would standardize the migration of keys from one HSM to another, using mechanisms already in the protocol or ones added for this purpose.
- Server to Server key management. This would extend the protocol to support communication between key management servers in different key management domains, for purposes of exporting and importing cryptographic material and potentially policy information.
- Multiple derived keys. This would allow the creation of multiple derived keys from one or more input keys. Note, however, that the current version of KMIP provides the capability to derive multiple keys and initialization vectors by creating a Secret Data object and specifying a cryptographic length equal to the total length of the derived objects.
- XML encoding. Expression of KMIP in XML rather than in tag/type/length/value may be considered for the future.
- Specification of Mask Generation Function. KMIP does not currently allow clients to specify the Mask Generation Function and assumes that encryption or signature schemes, such as OAEP or PSS, use MGF1 with the hash function as specified in the Cryptographic Parameters attribute. Client specification of MGFs may be considered for the future.
- Server monitoring of client status. This would enable the transfer of information about the client and its cryptographic module to the server. This information would enable the server to generate alarms and/or disallow requests from a client running component versions with known vulnerabilities.
- Symmetric key pairs. Only a subset of the cryptographic usage bits of the Cryptographic Usage Mask attribute may be permitted for keys distributed to a particular client. KMIP does not currently address how to securely assign and determine the applicable cryptographic usage for a client.
- Hardware-protected attribute. This attribute would allow clients and servers to determine if a key may only be processed inside a secure cryptographic device, such as an HSM. If this attribute is set, the key may only exist in cleartext within a secure hardware device, and all security-relevant attributes are bound to it in such a way that they may not be modified outside of such a secure device.
- Alternative profiles for key establishment. Less capable end-clients may not be able to support TLS and should use a proxy to communicate with the key management system. The KMIP protocol does not currently define alternative profiles, nor does it allow end-clients relying on the proxy model to securely establish a key with the server.

- Attribute mutation. The possibility for the server to use attribute values different than requested by the client if these values are not suitable for the server, and return these values in the response, instead of failing the request.
- Cryptographic Domain Parameters. KMIP allows a limited number of parameters to be specified during a Create Key Pair operation. Additional parameters may be considered for the future.
- Certificate Suspension/Unsuspension. KMIP does not currently support certificate suspension (putting a certificate on hold) or unsuspension (removing a certificate from hold). Adding support for certificate suspension/unsuspension into KMIP may be considered for the future.
- Namespace registration. Establishing a registry for namespaces may be considered for the future.
- Registering extensions to KMIP enumerations. Establishing a registry for extensions to defined KMIP enumerations, such as in support of profiles specific to IEEE P1619.3 or other organizations, may be considered for the future.

In addition to the functionality listed above, the KMIP TC is interested in establishing a C&A (certification and accreditation) process for independent validation of claims of KMIP conformance. Defining and establishing this process is a candidate for work by the KMIP TC after V1.1.

5 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:

Original Authors of the initial contribution:

David Babcock, HP
Steven Bade, IBM
Paolo Bezoari, NetApp
Mathias Björkqvist, IBM
Bruce Brinson, EMC
Christian Cachin, IBM
Tony Crossman, Thales/nCipher
Stan Feather, HP
Indra Fitzgerald, HP
Judy Furlong, EMC
Jon Geater, Thales/nCipher
Bob Griffin, EMC
Robert Haas, IBM
Timothy Hahn, IBM
Jack Harwood, EMC
Walt Hubis, LSI
Glen Jaquette, IBM
Jeff Kravitz, IBM
Michael McIntosh, IBM
Brian Metzger, HP
Anthony Nadalin, IBM
Elaine Palmer, IBM
Joe Pato, HP
René Pawlitzek, IBM
Subhash Sankuratripati, NetApp
Mark Schiller, HP
Martin Skagen, Brocade
Marcus Streets, Thales/nCipher
John Tattan, EMC
Karla Thomas, Brocade
Marko Vukolić , IBM
Steve Wierenga, HP

Participants in KMIP Usage Guide V1.1

Hal Aldridge, Sypris Electronics
Mike Allen, Symantec
Gordon Arnold, IBM
Todd Arnold, IBM
Matthew Ball, Oracle Corporation
Elaine Barker, NIST
Peter Bartok, Venafi, Inc.
Mathias Björkqvist, IBM
Kelley Burgin, National Security Agency
John Clark, Hewlett-Packard
Tom Clifford, Symantec Corp.
Graydon Dodson, Lexmark International Inc.
Chris Dunn, SafeNet, Inc.
Michael Duren, Sypris Electronics
Paul Earsy, SafeNet, Inc.

Stan Feather, Hewlett-Packard
Indra Fitzgerald, Hewlett-Packard
Alan Frindell, SafeNet, Inc.
Judith Furlong, EMC Corporation
Jonathan Geater, Thales e-Security
Susan Gleeson, Oracle
Robert Griffin, EMC Corporation
Paul Grojean, Individual
Robert Haas, IBM
Thomas Hardjono, M.I.T.
Steve He, Vormetric Inc.
Kurt Heberlein, Hewlett-Packard
Joel Hockey, Cryptsoft Pty Ltd.
Larry Hofer, Emulex Corporation
Brandon Hoff, Emulex Corporation
Walt Hubis, NetApp
Tim Hudson, Cryptsoft Pty Ltd.
Jay Jacobs, Target Corporation
Glen Jaquette, IBM
Scott Kipp, Brocade Communications Systems, Inc.
Kathy Kriese, Symantec Corporation
David Lawson, Emulex Corporation
John Leiseboer, Quintenssence Labs
Hal Lockhart, Oracle Corporation
Robert Lockhart, Thales e-Security
Anne Luk, Cryptsoft Pty Ltd.
Shyam Mankala, EMC Corporation
Upendra Mardikar, PayPal Inc.
Luther Martin, Voltage Security
Hyrum Mills, Mitre Corporation
Bob Nixon, Emulex Corporation
René Pawlitzek, IBM
John Peck, IBM
Rob Philpott, EMC Corporation
Denis Pochuev, SafeNet, Inc.
Ajai Puri, SafeNet Inc.
Peter Reed, SafeNet Inc.
Bruce Rich, IBM
Warren Robbins, Credant Systems
Saikat Saha, SafeNet, Inc.
Subhash Sankuratripati, NetApp
Mark Schiller, Hewlett-Packard
Brian Spector, Certivox
Terence Spies, Voltage Security
Marcus Streets, Thales e-Security
Kiran Thota, VMware
Sean Turner, IECA, Inc.
Paul Turner, Venafi, Inc.
Marko Vukolić, EURECOM
Rod Wideman, Quantum Corporation
Steven Wierenga, Hewlett-Packard
Peter Yee, EMC Corporation
Krishna Yellepeddy, IBM
Michael Yoder, Vormetric. Inc.
Peter Zelechoski, Election Systems & Software
Magda Zdunkiewicz, Cryptsoft

Appendix B. Acronyms

The following abbreviations and acronyms are used in this document:

3DES	- Triple Data Encryption Standard specified in ANSI X9.52
AES	- Advanced Encryption Standard specified in FIPS 197
ANSI	- American National Standards Institute
ARQC	- Authorization Request Cryptogram
ASCII	- American Standard Code for Information Interchange
CA	- Certification Authority
CBC	- Cipher Block Chaining specified in NIST SP 800-38A
CMC	- Certificate Management Messages over CMS specified in RFC 5275
CMP	- Certificate Management Protocol specified in RFC 4210
CRL	- Certificate Revocation List specified in RFC 5280
CRMF	- Certificate Request Message Format specified in RFC 4211
CVC	- Card Verification Code
DES	- Data Encryption Standard specified in FIPS 46-3
DEK	- Data Encryption Key
DH	- Diffie-Hellman specified in ANSI X9.42
FIPS	- Federal Information Processing Standard
GCM	- Galois/Counter Mode specified in NIST SP 800-38D
HMAC	- Keyed-Hash Message Authentication Code specified in FIPS 198-1
HSM	- Hardware Security Module
HTTP	- Hyper Text Transfer Protocol
HTTP(S)	- Hyper Text Transfer Protocol (Secure socket)
ID	- Identification
IP	- Internet Protocol
IPSec	- Internet Protocol Security
KEK	- Key Encryption Key
KMIP	- Key Management Interoperability Protocol
LTO4	- Linear Tape-Open, Generation 4
LTO5	- Linear Tape-Open, Generation 5
MAC	- Message Authentication Code
MD5	- Message Digest 5 Algorithm specified in RFC 1321
MGF	- Mask Generation Function
NIST	- National Institute of Standards and Technology
OAEP	- Optimal Asymmetric Encryption Padding specified in PKCS#1

1984 PEM - Privacy Enhanced Mail specified in RFC 1421
1985 PGP - OpenPGP specified in RFC 4880
1986 PKCS - Public-Key Cryptography Standards
1987 POP - Proof of Possession
1988 POSIX - Portable Operating System Interface
1989 PSS - Probabilistic Signature Scheme specified in PKCS#1
1990 RSA - Rivest, Shamir, Adelman (an algorithm)
1991 SHA - Secure Hash Algorithm specified in FIPS 180-2
1992 SP - Special Publication
1993 S/MIME - Secure/Multipurpose Internet Mail Extensions
1994 TCP - Transport Control Protocol
1995 TLS - Transport Layer Security
1996 TTLV - Tag, Type, Length, Value
1997 URI - Uniform Resource Identifier
1998 UTF-8 - Universal Transformation Format 8-bit specified in RFC 3629
1999 X.509 - Public Key Certificate specified in RFC 5280
2000 XML - Extensible Markup Language

2001 # Appendix C. Revision History

Revision	Date	Editor	Changes Made
wd-01	2011-07-26	Indra Fitzgerald	Incorporated the following proposals: Supporting Rekey of Asymmetric Key Pairs within KMIP, Encoding Options for Key Wrap, Discover Versions, Vendor Extensions, Cryptographic Length of Asymmetric Keys, and Text String Representation of Distinguished Names. Extended the multi-vendor interoperability method to include LTO5. Incorporated Usage Guide comments.
wd-02	2011-08-10	Indra Fitzgerald	Incorporated the Device Credential and Group proposal. Removed the Text String Representation of Distinguished Names section.
wd-03	2011-08-17	Indra Fitzgerald	Performed minor editorial changes.
wd-04	2011-10-7	Robert Griffin	Added new participant list and other minor editorial changes.
wd-05	2011-10-19	Robert Griffin	Converted to new OASIS template for non-standard-track document.
wd-06	2011-12-1	Robert Griffin	Incorporates new text from "v3KMIP1.1CertAttributeUpdateProposal.docx"
wd-07	2011-12-20	Robert Griffin	Editorial corrections to references and formatting.
cnd-01	2012-1-4	OASIS admin	Committee Note Draft for Public Review
wd-08	2012-4-4	Robert Griffin	Incorporate comments from public review
wd-09	2012-4-16	Robert Griffin	Incorporate comments from KMIP TC
wd-10	2012-4-26	Robert Griffin	Removed attribute index text, as voted by KMIP TC and updated contributors' list

2002