Key Management Interoperability Protocol Usage Guide Version 1.0

Committee Specification 01

15 June 2010

Specification URIs:

This Version:

http://docs.oasis-open.org/kmip/ug/v1.0/cs01/kmip-ug-1.0-cs-01.html
http://docs.oasis-open.org/kmip/ug/v1.0/cs01/kmip-ug-1.0-cs-01.doc (Authoritative)
http://docs.oasis-open.org/kmip/ug/v1.0/cs01/kmip-ug-1.0-cs-01.pdf

Previous Version:

http://docs.oasis-open.org/kmip/ug/v1.0/cd10/kmip-ug-1.0-cd-10.html
http://docs.oasis-open.org/kmip/ug/v1.0/cd10/kmip-ug-1.0-cd-10.doc (Authoritative)
http://docs.oasis-open.org/kmip/ug/v1.0/cd10/kmip-ug-1.0-cd-10.pdf

Latest Version:

http://docs.oasis-open.org/kmip/ug/v1.0/kmip-ug-1.0.html
http://docs.oasis-open.org/kmip/ug/v1.0/kmip-ug-1.0.doc
http://docs.oasis-open.org/kmip/ug/v1.0/kmip-ug-1.0.pdf

Technical Committee:

OASIS Key Management Interoperability Protocol (KMIP) TC

Chair(s):

Robert Griffin, EMC Corporation <robert.griffin@rsa.com>
Subhash Sankuratripati, NetApp <Subhash.Sankuratripati@netapp.com>

Editor(s):

Indra Fitzgerald, HP <indra.fitzgerald@hp.com>

Related work:

This specification replaces or supersedes:

- None

This specification is related to:

- Key Management Interoperability Protocol Specification Version 1.0
- Key Management Interoperability Protocol Profiles Version 1.0
- Key Management Interoperability Protocol Use Cases Version 1.0

Declared XML Namespace(s):

None

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.

Status:

This document was last revised or approved by the Key Management Interoperability Protocol TC on the above date. The level of approval is also listed above. Check the “Latest Version” or “Latest Approved Version” location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification 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/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/kmip/ipr.php.)

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/kmip/.

Notices

Table of Contents

Tables

1 Introduction

This Key Management Interoperability Protocol Usage Guide 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 V1.0, 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]**.

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

1.1 Terminology

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

1.2 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] OASIS Committee Specification 01, Key Management Interoperability Protocol Specification Version 1.0, June 2010, http://docs.oasis-open.org/kmip/spec/v1.0/cs01/kmip-spec-1.0-cs-01.doc

[KMIP-Prof] OASIS Committee Specification 01, Key Management Interoperability Protocol Profiles Version 1.0, June 2010, http://docs.oasis-open.org/kmip/profiles/v1.0/cs01/kmip-profiles-1.0-cs-01.doc

[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.

[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-UC] OASIS Committee Specification 01, Key Management Interoperability Protocol Use Cases Version 1.0, June 2010, http://docs.oasis-open.org/kmip/usecases/v1.0/cs01/kmip-usecases-1.0-cs-01.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. State-less server operation is much more reliable and easier to implement than stateful operation, and is consistent with possible implementation scenarios, such as web-services-based servers. This does not mean that the server itself maintains no state, only that the protocol does not require this.

2.4 Extensible Protocol

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

2.5 Server Policy

A server is required to be conformant to KMIP and support 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 all reasonable 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. 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 field 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 object, 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 object (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 object, the server interprets the identity of the requestor from the Credential object. 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 object, the server verifies that the usernames are the same. If they differ, the authentication fails and the server returns an error. If no Credential object 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 object 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 field is provided in the Message Request, the Password field is an optional field in the Username and Password structure of the Credential object.

The Credential object 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 object may be used in this scenario to identify individual clients by specifying the username in the Username and Password structure. Depending on the client's environment, the username may be the device's serial number, the volume name or some other unique identifier.

Multiple clients should not be authenticated using the same channel-level authentication credential (e.g., the same client certificate). The Credential object may be used to authenticate individual clients by requiring the Username and Password to be provided in the Credential object.

3.2 Authorization for Revoke, Recover, Destroy and Archive Operations

Neither authentication nor authorization is handled by the KMIP protocol directly. In particular, the Credential attribute is not guaranteed to be an authenticated identity of the requesting client. However, 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, with the single exception of the Query operation.

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 on key security. When a server receives a request for one of these operations, it should ensure that the client has authenticated its identity (see the Authentication Suites section in **[KMIP-Prof]**). The server should also ensure that the client requesting the operation is an object creator, 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.

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 and have, therefore, been included in **[KMIP-Spec]**.

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

- The client registers 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 properly authenticated to forestall man-in-the-middle attacks in which the client receives Notify or Put operations from an unauthenticated server. 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 Section 3 of **[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 negotiated by policy exchange between the client and server, outside the Key Management Interoperability Protocol.

  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.

- 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 is required to return an error, if the client invokes the Destroy operation on an Active object. To destroy an Active object, clients are required to 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
- 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, Derive Key, Certify, and Re-certify operations. All of these operations with the exception of the Create Key Pair operation use the Template-Attribute tag. The Create Key Pair operation uses 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 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; however, this particular example sets this attribute separately.

3.7 Archive Operations

When the Archive operation is performed, it is recommended that an object 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, on the other hand, 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 is intended to return 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 specified by RACF and JKS, 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 related operations. For example, Locate and Get are likely to be commonly accomplished within a single batch request.

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

3.16 Maximum Message Size

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

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

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

Both the Re-key and the Re-certify operations allow the specification of an offset interval.

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

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

3.18 Locate Queries

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

- Exact match of a transition to a given state. Locate the key(s) with a transition to a certain state at a specified time (t).
- Range match of a transition to a given state. Locate the key(s) with a transition to a certain state at any time at or between two specified times (t and t').
- Exact match of a state at a specified time. Locate the key(s) that are in a certain state at a specified time (t).

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

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

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

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

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

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

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

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
Obtain Lease	ID of Object	no change
Get Usage Allocation	ID of Key	no change
Check	ID of Object	no change

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

3.20 Key Block

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

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

- Unique Identifier – uniquely identifies the key
- Cryptographic Algorithm (e.g., AES, 3DES, RSA) – this attribute is either specified inside the Key Block structure or the Key Value structure
- Cryptographic Length (e.g., 128, 256, 2048) – this attribute is either specified inside the Key Block structure or the Key Value structure
- Cryptographic Usage Mask– identifies the cryptographic usage of the key (e.g., Encrypt, Wrap Key, Export)
- Cryptographic Parameters – provides additional parameters for determining how the key may be used
  - Block Cipher Mode (e.g., CBC, NISTKeyWrap, GCM) – this parameter identifies the mode of operation, including block cipher-based MACs or wrapping mechanisms
  - Padding Method (e.g., OAEP, X9.31, PSS) – identifies the padding method and if applicable the signature or encryption scheme
  - Hashing Algorithm (e.g., SHA-256) – identifies the hash algorithm to be used with the signature/encryption mechanism or Mask Generation Function; note that the different HMACs are defined individually as algorithms and do not require the Hashing Algorithm parameter to be set
  - Key Role Type – Identifies the 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 Wrapper 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 error.

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 currently 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-256)
- 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.

3.22 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 character 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.

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 Operation

The Create Key Pair operation allows 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:

<u>Common Template-Attribute</u>

- 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:

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

- 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 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 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 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 seem relevant only for 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 adequately support asymmetric concepts using symmetric keys. This is fairly 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.

Table 2: Cryptographic Usage Masks Pairs

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

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

3.28 Application Specific Information

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

- SMIME, 'someuser@company.com'
- TLS, 'some.domain.name'
- Volume Identification, '123343434'
- File Name, 'secret.doc'

- Client Generated Key ID, '450994003'

The following Application Name Spaces are recommended:

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

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 Name Space is recognized and supported by the server. An example for requesting the server to generate the Application Data is shown below:

AddAttribute(UID, AppSpecInfo{AppNameSpace='LIBRARY-LTO4'});

If the server does not recognize the name space, the "Application Name Space 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 Name Space 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(UID, 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 will be 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 Name Space is used to identify the namespace (such as LIBRARY-LTO4).
- 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 text format. 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).
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 Name Space 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 for all possible key descriptors when names are directly referenced.
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.
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

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 Name Space 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 7F65454323630384C343030303930353 23830343132<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

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

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 9.1.3.2.17 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, and, at this point of time, attribute certificates are considered 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]** 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 current KMIP Specification supports certificate renewals using the Re-Certify operation and certificate updates using the Certify operation. Support for certificate rekey is not currently supported by KMIP, since certificate rekey requires the ability to rekey an asymmetric key pair a capability not currently supported by KMIP. Support for rekey of asymmetric key pairs, along with certificate rekey, may be considered for a future KMIP release.

3.33 Key Encoding

Two parties receiving the same key as a Key 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, one will notice 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 for DSA, DH, ECDSA, etc. 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.

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.
- Certificate creation without client-provided Certificate Request. This would allow clients to request the server to perform the Certify or Re-certify operation from the specified key pair IDs without providing a Certificate Request.
- 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 support 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.
- Re-key support for other cryptographic objects. The Re-key operation is currently restricted to symmetric keys. Applying Re-key to other cryptographic objects, such as asymmetric keys and certificates, 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 V1.0.

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 on the cover page of this document.

A. 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
JKS - Java Key Store
KEK - Key Encryption Key
KMIP - Key Management Interoperability Protocol
LTO4 - Linear Tape-Open 4
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
PEM - Privacy Enhanced Mail specified in RFC 1421

PGP - OpenPGP specified in RFC 4880
PKCS - Public-Key Cryptography Standards
POP - Proof of Possession
POSIX - Portable Operating System Interface
PSS - Probabilistic Signature Scheme specified in PKCS#1
RACF - Remote Access Control Facility
RSA - Rivest, Shamir, Adelman (an algorithm)
SHA - Secure Hash Algorithm specified in FIPS 180-2
SP - Special Publication
S/MIME - Secure/Multipurpose Internet Mail Extensions
TCP - Transport Control Protocol
TLS - Transport Layer Security
TTLV - Tag, Type, Length, Value
URI - Uniform Resource Identifier
UTF-8 - Universal Transformation Format 8-bit specified in RFC 3629
X.509 - Public Key Certificate specified in RFC 5280
XML - Extensible Markup Language

B. 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:

Mike Allen, PGP Corporation
Gordon Arnold, IBM
Todd Arnold, IBM
Matthew Ball, Oracle Corporation
Elaine Barker, NIST
Peter Bartok, Venafi, Inc.
Mathias Björkqvist, IBM
Kevin Bocek, Thales e-Security
Kelley Burgin, National Security Agency
Jon Callas, PGP Corporation
Tom Clifford, Symantec Corp.
Graydon Dodson, Lexmark International Inc.
Chris Dunn, SafeNet, Inc.
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
Robert Griffin, EMC Corporation
Robert Haas, IBM
Thomas Hardjono, M.I.T.
Kurt Heberlein, 3PAR, Inc.
Marc Hocking, BeCrypt Ltd.
Larry Hofer, Emulex Corporation
Brandon Hoff, Emulex Corporation
Walt Hubis, LSI Corporation
Tim Hudson, Cryptsoft Pty Ltd.
Wyllys Ingersoll, Oracle Corporation
Jay Jacobs, Target Corporation
Glen Jaquette, IBM
Scott Kipp, Brocade Communications Systems, Inc.
David Lawson, Emulex Corporation
Hal Lockhart, Oracle Corporation
Robert Lockhart, Thales e-Security
Shyam Mankala, EMC Corporation
Upendra Mardikar, PayPal Inc.
Marc Massar, Individual
Don McAlister, Associate
Hyrum Mills, Mitre Corporation
Bob Nixon, Emulex Corporation
Landon Curt Noll, Cisco Systems, Inc.
René Pawlitzek, IBM
John Peck, IBM
Rob Philpott, EMC Corporation
Scott Rea, Individual
Bruce Rich, IBM
Scott Rotondo, Oracle Corporation
Saikat Saha, Vormetric, Inc.
Anil Saldhana, Red Hat
Subhash Sankuratripati, NetApp
Mark Schiller, Hewlett-Packard
Jitendra Singh, Brocade Communications Systems, Inc.
Servesh Singh, EMC Corporation
Terence Spies, Voltage Security
Sandy Stewart, Oracle Corporation
Marcus Streets, Thales e-Security
Brett Thompson, SafeNet, Inc.
Benjamin Tomhave, Individual
Sean Turner, IECA, Inc.
Paul Turner, Venafi, Inc.
Marko Vukolić, IBM
Rod Wideman, Quantum Corporation
Steven Wierenga, Hewlett-Packard
Peter Yee, EMC Corporation
Krishna Yellepeddy, IBM
Peter Zelechoski, Election Systems & Software
Grace Zhang, Skyworth TTG Holdings Limited

1492 # C. Revision History

Revision	Date	Editor	Changes Made
ed-0.98	2009-04-29	Indra Fitzgerald	Initial conversion of input document to OASIS format.
ed-0.98	2009-07-28	Indra Fitzgerald	Added clarifications, examples, and deferred items.
ed-0.98	2009-09-08	Indra Fitzgerald	Added approved proposals and incorporated Elaine Barker's comments.
ed-0.98	2009-09-23	Indra Fitzgerald	Removed KMIP Profiles section and incorporated the Interoperable Key Naming for Tape proposal.
ed-0.98	2009-09-24	Indra Fitzgerald	Removed the Conformance section; added additional Certificate Request and POP text to Certify and Re-certify; added the Revocation Reason Codes section.
draft-01	2009-10-07	Indra Fitzgerald	Incorporated the Certificate Renewal, Update, Re-key proposal, the Key Encoding proposal; removed normative words "must", "shall", "required", "will", and "can"; added Create Key Pair example; updated the references and acronyms list; incorporated comments from RobertH and SubhashS; updated the Authentication section; added minor edits and clarifications.
draft-02	2009-10-09	Indra Fitzgerald	Incorporated Rod Wideman's comments on the language. Changed the heading indentation, paragraph style, and list styles according to the OASIS template guidelines. Added additional references. Replaced the TBDs. Added a use-case for registering a wrapped key as an opaque cryptographic object.
draft-03	2009-10-21	Indra Fitzgerald	Added the list of participants to Appendix B. Clarified the Authentication section (section 3.1) and added examples. Modified the title page. Performed minor editorial changes.
draft-04	2009-11-06	Indra Fitzgerald	Incorporated Elaine's comments. This is the tentative revision for public review.
draft-05	2009-11-09	Indra Fitzgerald	Minor edits to the reference sections.
draft-06	2010-02-24	Indra Fitzgerald	Addressed public review comments. Clarified how templates work (section 3.6). Added Judy Furlong's proposal on using the same asymmetric key pair in multiple algorithms (section 3.34).
draft-07	2010-03-04	Indra Fitzgerald	Clarified that the Destroy operation cannot destroy Active objects (section 3.5).
draft-08	2010-03-17	Indra Fitzgerald	Added the Server Policy section (2.5). Added the Credential section (3.1.1) to the Authentication section. Replaced SSL/TLS with TLS. Updated the participant list. Other minor edits.
draft-09	2010-03-18	Indra Fitzgerald	Renamed Role Type to Key Role Type. Updated the participant list.

draft-10	2010-05-26	Indra Fitzgerald	Minor edits to the reference and acronym sections. Updated the participant list.