



# Key Management Interoperability Protocol Usage Guide Version 1.0

Committee Draft 09 / Public Review 02

18 March 2010

**Specification URIs:**

**This Version:**

<http://docs.oasis-open.org/kmip/ug/v1.0/cd09/kmip-ug-1.0-cd-09.html>  
<http://docs.oasis-open.org/kmip/ug/v1.0/cd09/kmip-ug-1.0-cd-09.doc> (Authoritative)  
<http://docs.oasis-open.org/kmip/ug/v1.0/cd09/kmip-ug-1.0-cd-09.pdf>

**Previous Version:**

<http://docs.oasis-open.org/kmip/ug/v1.0/cd05/kmip-ug-1.0-cd-05.html>  
<http://docs.oasis-open.org/kmip/ug/v1.0/cd05/kmip-ug-1.0-cd-05.doc>  
<http://docs.oasis-open.org/kmip/ug/v1.0/cd05/kmip-ug-1.0-cd-05.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](mailto:robert.griffin@rsa.com)>  
Subhash Sankuratripati, NetApp <[Subhash.Sankuratripati@netapp.com](mailto:Subhash.Sankuratripati@netapp.com)>

**Editor(s):**

Indra Fitzgerald, HP <[indra.fitzgerald@hp.com](mailto: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

Copyright © OASIS® 2010. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", "KMIP" are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

---

# Table of Contents

1	Introduction.....	6
1.1	Terminology.....	6
1.2	Normative References.....	6
1.3	Non-normative References.....	9
2	Assumptions.....	10
2.1	Island of Trust.....	10
2.2	Message Security.....	10
2.3	State-less Server.....	10
2.4	Extensible Protocol.....	10
2.5	Server Policy.....	10
2.6	Support for Cryptographic Objects.....	10
2.7	Client-Server Message-based Model.....	11
2.8	Synchronous and Asynchronous Messages.....	11
2.9	Support for “Intelligent Clients” and “Key Using Devices“.....	11
2.10	Batched Requests and Responses.....	11
2.11	Reliable Message Delivery.....	11
2.12	Large Responses.....	11
2.13	Key Life-cycle and Key State.....	11
3	Usage Guidelines.....	12
3.1	Authentication.....	12
3.1.1	Credential.....	12
3.2	Authorization for Revoke, Recover, Destroy and Archive Operations.....	13
3.3	Using Notify and Put Operations.....	13
3.4	Usage Allocation.....	13
3.5	Key State and Times.....	14
3.6	Template.....	15
3.6.1	Template Usage Examples.....	16
3.7	Archive Operations.....	17
3.8	Message Extensions.....	17
3.9	Unique Identifiers.....	17
3.10	Result Message Text.....	17
3.11	Query.....	17
3.12	Canceling Asynchronous Operations.....	17
3.13	Multi-instance Hash.....	17
3.14	Returning Related Objects.....	18
3.15	Reducing Multiple Requests through the Use of Batch.....	18
3.16	Maximum Message Size.....	18
3.17	Using Offset in Re-key and Re-certify Operations.....	18
3.18	Locate Queries.....	18
3.19	ID Placeholder.....	20
3.20	Key Block.....	20
3.21	Using Wrapped Keys with KMIP.....	21

3.21.1	Encrypt-only Example with a Symmetric Key as an Encryption Key for a Get Request and Response	22
3.21.2	Encrypt-only Example with a Symmetric Key as an Encryption Key for a Register Request and Response	22
3.21.3	Encrypt-only Example with an Asymmetric Key as an Encryption Key for a Get Request and Response	23
3.21.4	MAC-only Example with an HMAC Key as an Authentication Key for a Get Request and Response	23
3.21.5	Registering a Wrapped Key as an Opaque Cryptographic Object	24
3.22	Object Group	24
3.23	Certify and Re-certify	24
3.24	Specifying Attributes during a Create Key Pair Operation	25
3.24.1	Example of Specifying Attributes during the Create Key Pair Operation	25
3.25	Registering a Key Pair	26
3.26	Non-Cryptographic Objects	27
3.27	Asymmetric Concepts with Symmetric Keys	27
3.28	Application Specific Information	28
3.29	Mutating Attributes	29
3.30	Interoperable Key Naming for Tape	30
3.30.1	Native Tape Encryption by a KMIP Client	30
3.31	Revocation Reason Codes	34
3.32	Certificate Renewal, Update, and Re-key	34
3.33	Key Encoding	34
3.33.1	AES Key Encoding	34
3.33.2	Triple-DES Key Encoding	35
3.34	Using the Same Asymmetric Key Pair in Multiple Algorithms	35
4	Deferred KMIP Functionality	36
5	Implementation Conformance	38
A.	Acronyms	39
B.	Acknowledgements	41
C.	Revision History	43

## Tables

Table 1:	ID Placeholder Prior to and Resulting from a KMIP Operation	20
Table 2:	Cryptographic Usage Masks Pairs	28

---

# 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](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](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 Draft 10, *Key Management Interoperability Protocol Specification Version 1.0*, March 2010. <http://docs.oasis-open.org/kmip/spec/v1.0/cd10/kmip-spec-1.0-cd-10.doc>
- [KMIP-Prof]** OASIS Committee Draft 05, *Key Management Interoperability Protocol Profiles Version 1.0*, March 2010. <http://docs.oasis-open.org/kmip/profiles/v1.0/cd05/kmip-profiles-1.0-cd-05.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>

- 45 [PKCS#8] RSA Laboratories, PKCS#8 v1.2: Private-Key Information Syntax Standard,  
46 November 1, 1993, <http://www.rsa.com/rsalabs/node.asp?id=2130>
- 47 [PKCS#10] RSA Laboratories, PKCS #10 v1.7: Certification Request Syntax Standard, May  
48 26, 2000, <http://www.rsa.com/rsalabs/node.asp?id=2132>
- 49 [RFC1319] B. Kaliski, *The MD2 Message-Digest Algorithm*, IETF RFC 1319, Apr 1992,  
50 <http://www.ietf.org/rfc/rfc1319.txt>
- 51 [RFC1320] R. Rivest, *The MD4 Message-Digest Algorithm*, IETF RFC 1320, Apr 1992,  
52 <http://www.ietf.org/rfc/rfc1320.txt>
- 53 [RFC1321] R. Rivest, *The MD5 Message-Digest Algorithm*, IETF RFC 1321, Apr 1992,  
54 <http://www.ietf.org/rfc/rfc1321.txt>
- 55 [RFC1421] J. Linn, *Privacy Enhancement for Internet Electronic Mail: Part I: Message*  
56 *Encryption and Authentication Procedures*, IETF RFC 1421, Feb 1993,  
57 <http://www.ietf.org/rfc/rfc1421.txt>
- 58 [RFC1424] B. Kaliski, *Privacy Enhancement for Internet Electronic Mail: Part IV: Key*  
59 *Certification and Related Services*, IETF RFC 1424, February 1993,  
60 <http://www.ietf.org/rfc/rfc1424.txt>
- 61 [RFC2104] H. Krawczyk, M. Bellare, R. Canetti, *HMAC: Keyed-Hashing for Message*  
62 *Authentication*, IETF RFC 2104, Feb 1997, <http://www.ietf.org/rfc/rfc2104.txt>
- 63 [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,  
64 <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- 65 [RFC2898] B. Kaliski, *PKCS #5: Password-Based Cryptography Specification Version 2.0*,  
66 IETF RFC 2898, Sep 2000, <http://www.ietf.org/rfc/rfc2898.txt>
- 67 [RFC3394] J. Schaad, R. Housley, *Advanced Encryption Standard (AES) Key Wrap*  
68 *Algorithm*, IETF RFC 3394, Sep 2002, <http://www.ietf.org/rfc/rfc3394.txt>
- 69 [RFC3447] J. Jonsson, B. Kaliski, *Public-Key Cryptography Standards (PKCS) #1: RSA*  
70 *Cryptography Specifications Version 2.1*, IETF RFC 3447 Feb 2003,  
71 <http://www.ietf.org/rfc/rfc3447.txt>
- 72 [RFC3629] F. Yergeau, *UTF-8, a transformation format of ISO 10646*, IETF RFC 3629, Nov  
73 2003, <http://www.ietf.org/rfc/rfc3629.txt>
- 74 [RFC3647] S. Chokhani, W. Ford, R. Sabett, C. Merrill, and S. Wu, *RFC3647: Internet*  
75 *X.509 Public Key Infrastructure Certificate Policy and Certification Practices*  
76 *Framework*, November 2003, <http://www.ietf.org/rfc/rfc3647.txt>
- 77 [RFC4210] C. Adams, S. Farrell, T. Kause and T. Mononen, *RFC2510: Internet X.509*  
78 *Public Key Infrastructure Certificate Management Protocol (CMP)*, September  
79 2005, <http://www.ietf.org/rfc/rfc4210.txt>
- 80 [RFC4211] J. Schaad, *RFC 4211: Internet X.509 Public Key Infrastructure Certificate*  
81 *Request Message Format (CRMF)*, September 2005,  
82 <http://www.ietf.org/rfc/rfc4211.txt>
- 83 [RFC4868] S. Kelly, S. Frankel, *Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-*  
84 *512 with IPsec*, IETF RFC 4868, May 2007, <http://www.ietf.org/rfc/rfc4868.txt>
- 85 [RFC4949] R. Shirey, *RFC4949: Internet Security Glossary, Version 2*, August 2007,  
86 <http://www.ietf.org/rfc/rfc4949.txt>
- 87 [RFC5272] J. Schaad and M. Meyers, *RFC5272: Certificate Management over CMS (CMC)*,  
88 June 2008, <http://www.ietf.org/rfc/rfc5272.txt>
- 89 [RFC5280] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley and W. Polk, *RFC*  
90 *5280: Internet X.509 Public Key Infrastructure Certificate and Certificate*  
91 *Revocation List (CRL) Profile*, May 2008, <http://www.ietf.org/rfc/rfc5280.txt>
- 92 [RFC5649] R. Housley, *Advanced Encryption Standard (AES) Key Wrap with Padding*  
93 *Algorithm*, IETF RFC 5649, Aug 2009, <http://www.ietf.org/rfc/rfc5649.txt>
- 94 [SP800-38A] M. Dworkin, *Recommendation for Block Cipher Modes of Operation – Methods*  
95 *and Techniques*, NIST Special Publication 800-38A, Dec 2001,  
96 <http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf>



97 [SP800-38B] M. Dworkin, *Recommendation for Block Cipher Modes of Operation: The CMAC*  
98 *Mode for Authentication*, NIST Special Publication 800-38B, May 2005,  
99 [http://csrc.nist.gov/publications/nistpubs/800-38B/SP\\_800-38B.pdf](http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf)

100 [SP800-38C] M. Dworkin, *Recommendation for Block Cipher Modes of Operation: the CCM*  
101 *Mode for Authentication and Confidentiality*, NIST Special Publication 800-38C,  
102 May 2004, [http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-](http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf)  
103 [38C\\_updated-July20\\_2007.pdf](http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf)

104 [SP800-38D] M. Dworkin, *Recommendation for Block Cipher Modes of Operation:*  
105 *Galois/Counter Mode (GCM) and GMAC*, NIST Special Publication 800-38D, Nov  
106 2007, <http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf>

107 [SP800-38E] M. Dworkin, *Recommendation for Block Cipher Modes of Operation: The XTS-*  
108 *AES Mode for Confidentiality on Block-Oriented Storage Devices*, NIST Special  
109 Publication 800-38E, Jan 2010, [http://csrc.nist.gov/publications/nistpubs/800-](http://csrc.nist.gov/publications/nistpubs/800-38E/nist-sp-800-38E.pdf)  
110 [38E/nist-sp-800-38E.pdf](http://csrc.nist.gov/publications/nistpubs/800-38E/nist-sp-800-38E.pdf)

111 [SP800-56A] E. Barker, D. Johnson, and M. Smid, *Recommendation for Pair-Wise Key*  
112 *Establishment Schemes Using Discrete Logarithm Cryptography (Revised)*, NIST  
113 Special Publication 800-56A, March 2007,  
114 [http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A\\_Revision1\\_Mar08-](http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.pdf)  
115 [2007.pdf](http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.pdf)

116 [SP800-56B] E. Barker, L. Chen, A. Regenscheid, M. Smid, *Recommendation for Pair-Wise*  
117 *Key Establishment Schemes Using Integer Factorization Cryptography*, NIST  
118 Special Publication 800-56B, August 2009,  
119 <http://csrc.nist.gov/publications/nistpubs/800-56B/sp800-56B.pdf>

120 [SP800-57-1] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, *Recommendations for Key*  
121 *Management - Part 1: General (Revised)*, NIST Special Publication 800-57 part  
122 1, March 2007, [http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-](http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf)  
123 [revised2\\_Mar08-2007.pdf](http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf)

124 [SP800-67] W. Barker, *Recommendation for the Triple Data Encryption Algorithm (TDEA)*  
125 *Block Cipher*, NIST Special Publication 800-67, Version 1.1, Revised 19 May  
126 2008, <http://csrc.nist.gov/publications/nistpubs/800-67/SP800-67.pdf>

127 [SP800-108] L. Chen, *Recommendation for Key Derivation Using Pseudorandom Functions*  
128 *(Revised)*, NIST Special Publication 800-108, October 2009,  
129 <http://csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf>

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

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

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

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

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

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

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

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

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



150 **1.3 Non-normative References**

151 **[KMIP-UC]** OASIS Committee Draft 09, *Key Management Interoperability Protocol Use*  
152 *Cases Version 1.0*, March 2010. [http://docs.oasis-](http://docs.oasis-open.org/kmip/usecases/v1.0/cd09/kmip-usecases-1.0-cd-09.doc)  
153 [open.org/kmip/usecases/v1.0/cd09/kmip-usecases-1.0-cd-09.doc](http://docs.oasis-open.org/kmip/usecases/v1.0/cd09/kmip-usecases-1.0-cd-09.doc)

---

## 154 2 Assumptions

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

### 157 2.1 Island of Trust

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

### 162 2.2 Message Security

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

### 168 2.3 State-less Server

169 The protocol operates on the assumption that the server is state-less, which means that there is no  
170 concept of “sessions” inherent in the protocol. State-less server operation is much more reliable and  
171 easier to implement than stateful operation, and is consistent with possible implementation scenarios,  
172 such as web-services-based servers. This does not mean that the server itself maintains no state, only  
173 that the protocol does not require this.

### 174 2.4 Extensible Protocol

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

### 179 2.5 Server Policy

180 A server is required to be conformant to KMIP and support the conformance clauses as specified in  
181 **[KMIP-Spec]**. However, a server may refuse a server-supported operation or client-settable attribute if  
182 disallowed by the server policy.

### 183 2.6 Support for Cryptographic Objects

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

- 186 • Symmetric Keys
- 187 • Split (multi-part) Keys
- 188 • Asymmetric Key Pairs and their components
- 189 • Digital Certificates
- 190 • Derived Keys
- 191 • Secret Data
- 192 • Opaque (non-interpretable) cryptographic objects

## 193 **2.7 Client-Server Message-based Model**

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

## 202 **2.8 Synchronous and Asynchronous Messages**

203 The protocol allows two modes of operation. Synchronous (mandatory) operations are those in which a  
204 client sends a request and waits for a response from the server. Polled Asynchronous operations  
205 (optional) are those in which the client sends a request, the server responds with a “pending” status, and  
206 the client polls the server for the completed response and completion status. Server implementations may  
207 choose not to support the Polled Asynchronous feature of the protocol.

## 208 **2.9 Support for “Intelligent Clients” and “Key Using Devices”**

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

## 213 **2.10 Batched Requests and Responses**

214 The protocol contains a mechanism for sending batched requests and receiving the corresponding  
215 batched responses, to allow for higher throughput on operations that deal with a large number of entities,  
216 e. g., requesting dozens or hundreds of keys from a server at one time, and performing operations in a  
217 group. An option is provided to indicate whether to continue processing requests after an earlier request  
218 in the batch fails or to stop processing the remaining requests in the batch. Note that there is no option to  
219 treat an entire batch as atomic, that is, if a request in the batch fails, then preceding requests in the batch  
220 are not undone or rolled back (see Section 3.15). A special ID Placeholder (see Section 3.19) is provided  
221 in KMIP to allow related requests in a batch to be pipelined.

## 222 **2.11 Reliable Message Delivery**

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

## 225 **2.12 Large Responses**

226 For requests that could result in large responses, a mechanism in the protocol allows a client to specify in  
227 a request the maximum allowed size of a response. The server indicates in a response to such a request  
228 that the response would have been too large and, therefore, is not returned.

## 229 **2.13 Key Life-cycle and Key State**

230 **[KMIP-Spec]** describes the key life-cycle model, based on the NIST SP 800-57 key state definitions  
231 **[SP800-57-1]**, supported by the KMIP protocol. Particular implications of the key life-cycle model in terms  
232 of defining time-related attributes of objects are discussed in Section 3.5 below.

233

234

## 3 Usage Guidelines

235  
236

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

237

### 3.1 Authentication

238  
239  
240  
241  
242

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.

243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257

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.

258  
259  
260

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.

261  
262  
263

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.

264

#### 3.1.1 Credential

265  
266  
267  
268  
269  
270  
271

**[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.

272

273  
274  
275  
276  
277  
278

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.

279  
280  
281

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.

## 282 3.2 Authorization for Revoke, Recover, Destroy and Archive 283 Operations

284 Neither authentication nor authorization is handled by the KMIP protocol directly. In particular, the  
285 Credential attribute is not guaranteed to be an authenticated identity of the requesting client. However,  
286 the authentication suite, as specified in [KMIP-Prof], describes how the client identity is established for  
287 KMIP-compliant implementations. This authentication is performed for all KMIP operations, with the single  
288 exception of the Query operation.

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

## 298 3.3 Using Notify and Put Operations

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

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

- 305 • The client registers with the server, so that the server knows how to locate the client to which a  
306 Notify or Put is being sent and which events for the Notify are supported. However, such  
307 registration is outside the scope of the KMIP protocol. Registration also includes a specification of  
308 whether a given client supports Put and Notify, and what attributes may be included in a Put for a  
309 particular client.
- 310 • Communication between the client and the server is properly authenticated to forestall man-in-  
311 the-middle attacks in which the client receives Notify or Put operations from an unauthenticated  
312 server. Authentication for a particular client/server implementation is at a minimum accomplished  
313 using one of the mandatory authentication mechanisms (see [KMIP-Prof]). Further strengthening  
314 of the client/server communications integrity by means of signed message content and/or  
315 wrapped keys is recommended. Attribute values other than “Last Change Date” should not be  
316 included in a Notify to minimize risk of exposure of attribute information.
- 317 • In order to minimize possible divergence of key or state information between client and server as  
318 a result of server-initiated communication, any client receiving Notify or Put messages returns  
319 acknowledgements of these messages to the server. This acknowledgement may be at  
320 communication layers below the KMIP layer, such as by using transport-level acknowledgement  
321 provided in TCP/IP.
- 322 • For client devices that are incapable of responding to messages from the server, communication  
323 with the server happens via a proxy entity that communicates with the server, using KMIP, on  
324 behalf of the client. It is possible to secure communication between a proxy entity and the client  
325 using other, potentially proprietary mechanisms.

## 326 3.4 Usage Allocation

327 Usage should be allocated and handled carefully at the client, since power outages or other types of  
328 client failures (crashes) may render allocated usage lost. For example, in the case of a key being used for  
329 the encryption of tapes, such a loss of the usage allocation information following a client failure during  
330 encryption may result in the necessity for the entire tape backup session to be re-encrypted using a

331 different key, if the server is not able to allocate more usage. It is possible to address this through such  
332 approaches as caching usage allocation information on stable storage at the client, and/or having  
333 conservative allocation policies at the server (e.g., by keeping the maximum possible usage allocation per  
334 client request moderate). In general, usage allocations should be as small as possible; it is preferable to  
335 use multiple smaller allocation requests rather than a single larger request to minimize the likelihood of  
336 unused allocation.

### 337 3.5 Key State and Times

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

- 339 • Initial Date: The date and time when the managed cryptographic object was first created by or  
340 registered at the server
- 341 • Activation Date: The date and time when the managed cryptographic object may begin to be used  
342 for applying cryptographic protection to data
- 343 • Process Start Date: The date and time when a managed symmetric key object may begin to be  
344 used for processing cryptographically protected data
- 345 • Protect Stop Date: The date and time when a managed symmetric key object may no longer be  
346 used for applying cryptographic protection to data
- 347 • Deactivation Date: The date and time when the managed cryptographic object may no longer be  
348 used for any purpose, except for decryption, signature verification, or unwrapping, but only under  
349 extraordinary circumstances and when special permission is granted
- 350 • Destroy Date: The date and time when the managed cryptographic object was destroyed
- 351 • Compromise Occurrence Date: The date and time when the managed cryptographic object was  
352 first believed to be compromised
- 353 • Compromise Date: The date and time when the managed cryptographic object is entered into the  
354 compromised state
- 355 • Archive Date: The date and time when the managed object was placed in Off-Line storage

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

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

- 360 • As discussed for each of these attributes in Section 3 of **[KMIP-Spec]**, a number of these times  
361 are set once and it is not possible for the client or server to modify them. However, several of the  
362 time attributes (particularly the Activation Date, Protect Start Date, Process Stop Date and  
363 Deactivation Date) may be set by the server and/or requested by the client. Coordination of time-  
364 related attributes between client and server, therefore, is primarily the responsibility of the server,  
365 as it manages the cryptographic object and its state. However, special conditions related to time-  
366 related attributes, governing when the server accepts client modifications to time-related  
367 attributes, may be negotiated by policy exchange between the client and server, outside the Key  
368 Management Interoperability Protocol.

369  
370 In general, state transitions occur as a result of operational requests, such as Create, Create Key  
371 Pair, Register, Activate, Revoke, and Destroy. However, clients may need to specify times in the  
372 future for such things as Activation Date, Deactivation Date, Process Start Date, and Protect Stop  
373 Date.

374  
375 KMIP allows clients to specify times in the past for such attributes as Activation Date and  
376 Deactivation Date. This is intended primarily for clients that were disconnected from the server at  
377 the time that the client performed that operation on a given key.

- 378 • It is valid to have a projected Deactivation Date when there is no Activation Date. This means,  
379 however, that the key is not yet active, even though its projected Deactivation Date has been  
380 specified. A valid Deactivation Date is greater than or equal to the Activation Date.



- 381 • The Protect Stop Date may be equal to, but may not be later than the Deactivation Date.  
382 Similarly, the Process Start Date may be equal to, but may not precede, the Activation Date.  
383 KMIP implementations should consider specifying both these attributes, particularly for symmetric  
384 keys, as a key may be needed for processing protected data (e.g., decryption) long after it is no  
385 longer appropriate to use it for applying cryptographic protection to data (e.g., encryption).
- 386 • KMIP does not allow an Active object to be destroyed with the Destroy operation. The server is  
387 required to return an error, if the client invokes the Destroy operation on an Active object. To  
388 destroy an Active object, clients are required to first call the Revoke operation or explicitly set the  
389 Deactivation Date of the object. Once the object is in Deactivated state, clients may destroy the  
390 object by calling the Destroy operation. These operations may be performed in a batch. If other  
391 time-related attributes (e.g., Protect Stop Date) are set to a future date, the server should set  
392 these to the Deactivation Date.
- 393 • After a cryptographic object is destroyed, a key management server may retain certain  
394 information about the object, such as the Unique Identifier.

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

## 402 3.6 Template

403 The usage of templates is an alternative approach for setting attributes in an operation request. Instead of  
404 individually specifying each attribute, a template may be used to set any of the following attributes for a  
405 managed object:

- 406 • Cryptographic Algorithm
- 407 • Cryptographic Length
- 408 • Cryptographic Domain Parameters
- 409 • Cryptographic Parameters
- 410 • Operation Policy Name
- 411 • Cryptographic Usage Mask
- 412 • Usage Limits
- 413 • Activation Date
- 414 • Process Start Date
- 415 • Protect Stop Date
- 416 • Deactivation Date
- 417 • Object Group
- 418 • Application Specific Information
- 419 • Contact Information
- 420 • Custom Attribute

421 In addition to these attributes, the template has attributes that are applicable to the template itself. These  
422 include the attributes (Unique Identifier, Initial Date, Last Change Date, and Archive Date) set implicitly  
423 after successfully completing a certain operation and attributes set by the client (Object Type and Name)  
424 in the Register request. When registering a template, the Name attribute for the template should be set. It  
425 is used to specify and identify the template in the Template-Attribute structure when attributes for a  
426 managed object are set.

427

428 The Template-Attribute structure allows for multiple template names and individual attributes to be  
429 specified in an operation request. The structure is used in the Create, Create Key Pair, Register, Re-key,  
430 Derive Key, Certify, and Re-certify operations. All of these operations with the exception of the Create  
431 Key Pair operation use the Template-Attribute tag. The Create Key Pair operation uses the Common  
432 Template-Attribute, Private Key Template Attribute, and Public Key Template-Attribute tags.

433

434 Templates may be the subject of the Register, Locate, Get, Get Attributes, Get Attribute List, Add  
435 Attribute, Modify Attribute, Delete Attribute, Delete Attribute, and Destroy operations. Clients are not able  
436 to create a template with the Create operation; instead templates are created using the Register  
437 operation. When the template is the subject of the operation, the Unique ID is used to identify the  
438 template. The template name is only used to identify the template inside a Template-Attribute structure.

### 439 **3.6.1 Template Usage Examples**

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

#### 443 **3.6.1.1 Example of Registering a Template**

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

446

447 The following is specified inside the Register Request Payload:

- 448 • Object Type: Template
- 449 • Template-Attribute:
  - 450 – Name: Template1
  - 451 – Cryptographic Algorithm: AES
  - 452 – Cryptographic Length: 256
  - 453 – Cryptographic Usage Mask: Encrypt and Decrypt
  - 454 – Operation Policy Name: OperationPolicy1

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

#### 459 **3.6.1.2 Example of Creating a Symmetric Key using a Template**

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

461

462 The following is specified in the Create Request Payload:

463

- 464 • Object Type: Symmetric Key
- 465 • Template-Attribute:
  - 466 – Name: Template1
  - 467 – Attribute:
    - 468 Name: AESkey
    - 469 Custom Attribute: x-ID74592

470

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

### 476 **3.7 Archive Operations**

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

### 480 **3.8 Message Extensions**

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

### 483 **3.9 Unique Identifiers**

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

### 486 **3.10 Result Message Text**

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

### 493 **3.11 Query**

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

### 496 **3.12 Canceling Asynchronous Operations**

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

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

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

### 506 **3.13 Multi-instance Hash**

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

### 513 **3.14 Returning Related Objects**

514 The key block is intended to return a single object, with associated attributes and other data. For those  
515 cases in which multiple related objects are needed by a client, such as the private key and the related  
516 certificate specified by RACF and JKS, the client should issue multiple Get requests to obtain these  
517 related objects.

### 518 **3.15 Reducing Multiple Requests through the Use of Batch**

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

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

### 529 **3.16 Maximum Message Size**

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

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

### 537 **3.17 Using Offset in Re-key and Re-certify Operations**

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

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

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

### 548 **3.18 Locate Queries**

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

- 550 • Exact match of a transition to a given state. Locate the key(s) with a transition to a certain state at  
551 a specified time (t).
- 552 • Range match of a transition to a given state. Locate the key(s) with a transition to a certain state  
553 at any time at or between two specified times (t and t’).
- 554 • Exact match of a state at a specified time. Locate the key(s) that are in a certain state at a  
555 specified time (t).

556 • Match of a state during an entire time range. Locate the key(s) that are in a certain state during  
557 an entire time specified with times (t and t'). Note that the Activation Date could occur at or before  
558 t and that the Deactivation Date could occur at or after t'+1.

559 • Match of a state at some point during a time range. Locate the key(s) that are in a certain state at  
560 some time at or between two specified times (t and t'). In this case, the transition to that state  
561 could be before the start of the specified time range.

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

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

566 • Exact match of a transition to a given state: Locate (ActivationDate(t)). Locate keys with an  
567 Activation Date of t.

568 • Range match of a transition to a given state: Locate (ActivationDate(t), ActivationDate(t')). Locate  
569 keys with an Activation Date at or between t and t'.

570 • Exact match of a state at a specified time: Locate (ActivationDate(0), ActivationDate(t),  
571 DeactivationDate(t+1), DeactivationDate(MAX\_INT), CompromiseDate(t+1),  
572 CompromiseDate(MAX\_INT)). Locate keys in the Active state at time t, by looking for keys with a  
573 transition to Active before or until t, and a transition to Deactivated or Compromised after t  
574 (because we don't want the keys that have a transition to Deactivated or Compromised before t).  
575 The server assumes that keys without a DeactivationDate or CompromiseDate is equivalent to  
576 MAX\_INT (i.e., infinite).

577 • Match of a state during an entire time range: Locate (ActivationDate(0), ActivationDate(t),  
578 DeactivationDate(t'+1), DeactivationDate(MAX\_INT), CompromiseDate(t'+1),  
579 CompromiseDate(MAX\_INT)). Locate keys in the Active state during the entire time from t to t'.

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

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

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

587 • Exact match of a transition to a given state: Locate (CompromiseDate(t), State(Destroyed-  
588 Compromised)) and Locate (DestroyDate(t), State(Destroyed-Compromised)). KMIP does not  
589 support the OR in the Locate request, so two requests should be issued. Locate keys that were  
590 Destroyed and transitioned to the Destroyed-Compromised state at time t, and locate keys that  
591 were Compromised and transitioned to the Destroyed-Compromised state at time t.

592 • Range match of a transition to a given state: Locate (CompromiseDate(t), CompromiseDate(t'),  
593 State(Destroyed-Compromised)) and Locate (DestroyDate(t), DestroyDate(t'), State(Destroyed-  
594 Compromised)). Locate keys that are Destroyed-Compromised and were Compromised or  
595 Destroyed at or between t and t'.

596 • Exact match of a state at a specified time: Locate (CompromiseDate(0), CompromiseDate(t),  
597 DestroyDate(0), DestroyDate(t)); nothing else is needed, since there is no exit transition. Locate  
598 keys with a Compromise Date at or before t, and with a Destroy Date at or before t. These keys  
599 are, therefore, in the Destroyed-Compromised state at time t.

600 • Match of a state during an entire time range: Locate (CompromiseDate(0), CompromiseDate(t),  
601 DestroyDate(0), DestroyDate(t)). Same as above. As there is no exit transition from the  
602 Destroyed-Compromised state, the end of the range (t') is irrelevant.

603 • Match of a state at some point during a time range: Locate (CompromiseDate(0),  
604 CompromiseDate(t'-1), DestroyDate(0), DestroyDate(t'-1)). Locate keys with a Compromise Date  
605 at or before t'-1, and with a Destroy Date at or before t'-1. As there is no exit transition from the  
606 Destroyed-Compromised state, the start of the range (t) is irrelevant.

607 **3.19 ID Placeholder**

608 A number of operations are affected by a mechanism referred to as the ID Placeholder. This is a  
 609 temporary variable consisting of a single Unique Identifier that is stored inside the server for the duration  
 610 of executing a batch of operations. The ID Placeholder is obtained from the Unique Identifier returned by  
 611 certain operations; the applicable operations are identified in Table 1, along with a list of operations that  
 612 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

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

614 **3.20 Key Block**

615 The protocol uses the Key Block structure to transport a key to the client or server. This Key Block  
 616 consists of the Key Value Type, the Key Value, and the Key Wrapping Data. The Key Value Type  
 617 identifies the format of the Key Material, e.g., Raw format or Transparent Key structure. The Key Value



618 consists of the Key Material and optional attributes. The Key Wrapping Data provides information about  
619 the wrapping key and the wrapping mechanism, and is returned only if the client requests the Key Value  
620 to be wrapped by specifying the Key Wrapping Specification inside the Get Request Payload. The Key  
621 Wrapping Data may also be included inside the Key Block if the client registers a wrapped key.

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

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

### 647 **3.21 Using Wrapped Keys with KMIP**

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

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

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

665 Prior to wrapping or unwrapping a key, the server should verify that the wrapping key is allowed to be  
666 used for the specified purpose. For example, if the Unique ID of a symmetric key is specified in the Key

667 Wrapping Specification inside the Get request, the symmetric key should have the “Wrap Key” bit set in  
668 its Cryptographic Usage Mask. Similarly, if the client registers a signed key, the server should verify that  
669 the Signature Key, as specified by the client inside the Key Wrapper Data, has the “Verify” bit set in the  
670 Cryptographic Usage Mask. If the wrapping key is not permitted to be used for the requested purpose  
671 (e.g., when the Cryptographic Usage Mask is not set), the server should return the Operation Failed error.

### 672 **3.21.1 Encrypt-only Example with a Symmetric Key as an Encryption Key** 673 **for a Get Request and Response**

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

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

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

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

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

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

### 699 **3.21.2 Encrypt-only Example with a Symmetric Key as an Encryption Key** 700 **for a Register Request and Response**

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

708 Similar to the example in 3.21.1 the key is wrapped using the AES key wrap. The Key Value includes four  
709 attributes: Cryptographic Algorithm, Cryptographic Length, Cryptographic Parameters, and Cryptographic  
710 Usage Mask.

711 The Key Wrapping Data includes the following information:

- 712 • Wrapping Method: Encrypt
- 713 • Encryption Key Information
  - 714 – Unique Key ID: Key ID of the AES wrapping key

715           – Cryptographic Parameters: The Block Cipher Mode is NISTKeyWrap (not necessary if default  
716           block cipher mode for wrapping key is NISTKeyWrap)

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

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

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

### 728 **3.21.3 Encrypt-only Example with an Asymmetric Key as an Encryption** 729 **Key for a Get Request and Response**

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

- 735       • Wrapping Method: Encrypt
- 736       • Encryption Key Information
  - 737           – Unique Key ID: Key ID of the RSA public key
  - 738           – Cryptographic Parameters:
    - 739               Padding Method: OAEP
    - 740               Hashing Algorithm: SHA-256

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

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

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

### 751 **3.21.4 MAC-only Example with an HMAC Key as an Authentication Key for** 752 **a Get Request and Response**

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

- 756       • Wrapping Method: MAC/sign
- 757       • MAC/Signature Key Information
  - 758           – Unique Key ID: Key ID of the MACing key (note that the algorithm associated with this key  
759           would be HMAC-256)
- 760       • Attribute Name: x-Nonce

761 For HMAC, no Cryptographic Parameters need to be specified, since the algorithm, including the hash  
762 function, may be determined from the Algorithm attribute set for the specified MAC Key. The server  
763 should verify that the HMAC key has the “MAC Generate” bit set in the Cryptographic Usage Mask. Note  
764 that an HMAC key does not require the “Wrap Key” bit to be set in the Cryptographic Usage Mask.

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

- 768 • Wrapping Method: MAC/sign
- 769 • MAC/Signature Key Information
- 770 • Unique Key ID: Key ID of the MACing key
- 771 • MAC/Signature: HMAC result of the Key Value

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

### 779 **3.21.5 Registering a Wrapped Key as an Opaque Cryptographic Object**

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

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

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

790 The Key Wrapping Data does not need to be specified.

## 791 **3.22 Object Group**

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

## 800 **3.23 Certify and Re-certify**

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

806 **[KMIP-Spec]** supports multiple options for submitting a certificate request to the key management server  
807 within a Certify or Re-Certify operation. It is a vendor decision as to whether the key management server

808 offers certification authority (CA) functionality or proxies the certificate request onto a separate CA for  
809 processing. The type of certificate request formats supported is also a vendor decision, and this may, in  
810 part, be based upon the request formats supported by any CA to which the server proxies the certificate  
811 requests.

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

## 820 **3.24 Specifying Attributes during a Create Key Pair Operation**

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

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

### 831 **3.24.1 Example of Specifying Attributes during the Create Key Pair** 832 **Operation**

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

#### 835 Common Template-Attribute

- 836 • RSACom Template
  - 837 – Cryptographic Algorithm: RSA
  - 838 – Cryptographic Length: 2048
  - 839 – Cryptographic Parameters: Padding Method OAEP
  - 840 – Custom Attribute: x-Serial 1234
  - 841 – Object Group: Key encryption group 1
- 842 • Attribute
  - 843 – Cryptographic Length: 4096
  - 844 – Cryptographic Parameters: Padding Method PKCS1 v1.5
  - 845 – Custom Attribute: x-ID 56789

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

#### 848 Private Key Template-Attribute

- 849 • RSAPriv Template
  - 850 – Object Group: Key encryption group 2
- 851 • Attribute



- 852           – Cryptographic Usage Mask: Unwrap Key
- 853           – Name: PrivateKey1

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

855 Public Key Template-Attribute

- 856           • Attribute
  - 857           – Cryptographic Usage Mask: Wrap Key
  - 858           – Name: PublicKey1

859  
860 Following the attribute precedence rule, the server creates a 4096-bit RSA key. The following client-  
861 specified attributes are set:

862 Private Key

- 863           • Cryptographic Algorithm: RSA
- 864           • Cryptographic Length: 4096
- 865           • Cryptographic Parameters: OAEP
- 866           • Cryptographic Parameters: PKCS1 v1.5
- 867           • Cryptographic Usage Mask: Unwrap Key
- 868           • Custom Attribute: x-Serial 1234
- 869           • Custom Attribute: x-ID 56789
- 870           • Object Group: Key encryption group 1
- 871           • Object Group: Key encryption group 2
- 872           • Name: PrivateKey1

873 Public Key

- 874           • Cryptographic Algorithm: RSA
- 875           • Cryptographic Length: 4096
- 876           • Cryptographic Parameters: OAEP
- 877           • Cryptographic Parameters: PKCS1 v1.5
- 878           • Cryptographic Usage Mask: Wrap Key
- 879           • Custom Attribute: x-Serial 1234
- 880           • Custom Attribute: x-ID 56789
- 881           • Object Group: Key encryption group 1
- 882           • Name: PublicKey1

### 883 **3.25 Registering a Key Pair**

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

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

- 893 1. Register the public key and set all associated attributes:
  - 894           a. Cryptographic Algorithm



- 895           b. Cryptographic Length
- 896           c. Cryptographic Usage Mask
- 897   2. Register the private key and set all associated attributes
- 898           a. Cryptographic Algorithm is the same for both public and private key
- 899           b. Cryptographic Length is the same for both public and private key
- 900           c. Cryptographic Parameters may be set; if set, the value is the same for both the public and  
901           private key
- 902           d. Cryptographic Usage Mask is set, but does not contain the same value for both the public  
903           and private key
- 904           e. Link is set with Link Type *Public Key Link* and the Linked Object Identifier of the  
905           corresponding Public Key
- 906           f. Link is set for the Public Key with Link Type *Private Key Link* and the Linked Object Identifier  
907           of the corresponding Private Key

### 908   **3.26 Non-Cryptographic Objects**

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

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

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

- 916       • Unique Identifier
- 917       • Object Type
- 918       • Digest
- 919       • State
- 920       • Initial Date
- 921       • Last Change Date

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

- 924       • Cryptographic Usage Mask

### 925   **3.27 Asymmetric Concepts with Symmetric Keys**

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

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

- 933       • In some of these operations, such as MAC generate and verify, the same cryptographic primitive  
934       is used in both of the complementary operations. MAC generation involves computing and  
935       returning the MAC, while MAC verification involves computing that same MAC and comparing it  
936       to a supplied value to determine if they are the same. Thus, both generation and verification use  
937       the “encrypt” operation, and the two usages are not able to be distinguished by considering only  
938       “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 <i>role</i> 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. <i>TRANSLATE ENCRYPT/DECRYPT</i> is used for data encipherment. <i>TRANSLATE WRAP/UNWRAP</i> 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'

965       • Client Generated Key ID, '450994003'

966 The following Application Name Spaces are recommended:

- 967       • SMIME
- 968       • TLS
- 969       • IPSEC
- 970       • HTTPS
- 971       • PGP
- 972       • Volume Identification
- 973       • File Name
- 974       • LTO4
- 975       • LIBRARY-LTO4

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

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

983 If the server does not recognize the name space, the “Application Name Space Not Supported” error is  
984 returned to the client.

985 If the Application Data is set to null, as shown in the example below, and the Application Name Space is  
986 recognized by the server, the server does not generate the Application Data for the client. The server  
987 stores the Application Specific Information attribute with the Application Data value set to null.

```
988       AddAttribute(UID, AppSpecInfo{AppNameSpace='LIBRARY-LTO4', AppData=null});
```

### 989 **3.29 Mutating Attributes**

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

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

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

999 If a server does not support backdated attributes, and cryptographic objects are expected to change state  
1000 at the specified current date and time (as perceived by the client), clients are recommended to issue the  
1001 operation that would implicitly set the date for the client. For example, instead of explicitly setting the  
1002 Activation Date, clients could issue the Activate operation. This would require the server to set the  
1003 Activation Date to the current date and time as perceived by the server.

1004 If it is not possible to set a date attribute via an operation, and the server does not support backdated  
1005 attributes, clients need to take into account that potential clock skew issues may cause the server to  
1006 return an error even if a date attribute is set to the client’s current date and time.

1007 For additional information, refer to the sections describing the State attribute and the Time Stamp field in  
1008 **[KMIP-Spec]**.

## 1009 **3.30 Interoperable Key Naming for Tape**

1010 This section describes methods for creating and storing key identifiers that are interoperable across multi-  
1011 vendor KMIP clients.

### 1012 **3.30.1 Native Tape Encryption by a KMIP Client**

1013 This method is primarily intended to promote interoperable key naming between tape library products  
1014 which already support non-KMIP key managers, where KMIP support is being added.

1015 When those existing library products become KMIP clients, a common method for naming and storing  
1016 keys may be used to support moving tape cartridges between the libraries, and successfully retrieving  
1017 keys, assuming that the clients have appropriate access privileges. The library clients may be from  
1018 multiple vendors, and may be served by a KMIP key manager from a different vendor.

#### 1019 **3.30.1.1 Method Overview**

- 1020 • The method uses the KMIP Application Specific Information (ASI) attribute's Application Data field  
1021 to store the key name. The ASI Application Name Space is used to identify the namespace (such  
1022 as LIBRARY-LTO4).
- 1023 • The method also uses the tape format's Key Associated Data (KAD) fields to store the key name.  
1024 Tape formats may provide both authenticated and unauthenticated storage for the KAD data. This  
1025 method ensures optimum utilization of the authenticated KAD data when the tape format supports  
1026 authentication.
- 1027 • The method supports both client-generated and server-generated key names.
- 1028 • The method, in many cases, is backward-compatible if tapes are returned to a non-KMIP key  
1029 manager environment.
- 1030 • Key names stored in the KMIP server's ASI attribute are always text format. Key names stored on  
1031 the KMIP client's KAD fields are always numeric format, due to space limitations of the tape  
1032 format. The method basically consists of implementing a specific algorithm for converting  
1033 between text and numeric formats.
- 1034 • The algorithm used by this conversion is reversible.

#### 1035 **3.30.1.2 Definitions**

- 1036 • Key Associated Data (KAD). Part of the tape format. May be segmented into authenticated and  
1037 unauthenticated fields. KAD usage is detailed in the SCSI SSC-3 standard from the T10  
1038 organization.
- 1039 • Application Specific Information (ASI). A KMIP attribute.
- 1040 • Hexadecimal numeric characters. Case-sensitive, printable, single byte ASCII characters  
1041 representing the numbers 0 through 9 and uppercase alpha A through F. (US-ASCII characters  
1042 30h-39h and 41h-46h).  
1043 Hexadecimal numeric characters are always paired, each pair representing a single 8-bit numeric  
1044 value. A leading zero character is provided, if necessary, so that every byte in the tape's KAD is  
1045 represented by exactly 2 hexadecimal numeric characters.  
1046
- 1047 • N(k). The number of bytes in the tape format's combined KAD fields (both authenticated and  
1048 unauthenticated).
- 1049 • N(a), N(u). The number of bytes in the tape format's authenticated, and unauthenticated KAD  
1050 fields, respectively.

#### 1051 **3.30.1.3 Algorithm 1. Numeric to text direction (tape format's KAD to KMIP ASI)**

1052 Description: All information contained in the tape format's KAD fields is converted to a null-terminated  
1053 ASCII string consisting of hexadecimal numeric character pairs. First, the unauthenticated KAD data is

1054 converted to text. Then, the authenticated KAD data is converted and appended to the end of the string.  
1055 The string is then null-terminated.

1056

1057 Implementation Example:

- 1058 1. Define an input buffer sized for  $N(k)$ . For LTO4,  $N(k)$  is 44 bytes (12 bytes authenticated, 32  
1059 unauthenticated).
- 1060 2. Define an output buffer sufficient to contain a null-terminated string with a maximum length of  
1061  $2*N(k)+1$  bytes.
- 1062 3. Define the standard POSIX (also known as C) locale. Each character in the string is a single-byte US-  
1063 ASCII character.
- 1064 4. Copy the tape format's KAD data, from the unauthenticated KAD field first, to the input buffer.  
1065 Effectively, the first byte (byte 0) of the input buffer is the first byte of unauthenticated KAD. Bytes  
1066 from the authenticated KAD are concatenated, after the unauthenticated bytes.
- 1067 5. For each byte in the input buffer, convert to US-ASCII as follows:
  - 1068 a. Convert the byte's value to exactly 2 hexadecimal numeric characters, including a leading 0  
1069 where necessary. Append these 2 numeric characters to the output buffer, with the high-nibble  
1070 represented by the left-most hexadecimal numeric character.
  - 1071 b. After all byte values have been converted, null terminate the output buffer.
- 1072 6. When storing the string to the KMIP server, use the object's ASI attribute's Application Data field.  
1073 Store the namespace (such as LIBRARY-LTO4) in the ASI attribute's Application Name Space field.

#### 1074 **3.30.1.4 Algorithm 2. Text to numeric direction (KMIP ASI to tape format's KAD)**

1075 Description: Hexadecimal numeric character pairs in the null-terminated ASCII string are converted to  
1076 single byte numeric values, and stored in the tape format's KAD fields. The authenticated KAD field is  
1077 populated first, from a sub-string consisting of the last  $2*N(a)$  characters in the full string. Any remaining  
1078 characters in the string are converted and stored to the unauthenticated KAD field. The null termination  
1079 byte is not converted.

1080

1081 Implementation Example:

- 1082 1. Obtain the key's name from the KMIP server's ASI attribute for that object. Copy the null terminated  
1083 string to an input buffer of size  $2*N(k) + 1$  bytes. For LTO4, an 89 character string, including null  
1084 termination, is sufficient for all possible key descriptors when names are directly referenced.
- 1085 2. Define output buffers for unauthenticated KAD, and authenticated KAD, of size  $N(u)$  and  $N(a)$   
1086 respectively. For LTO4, this would be 32 bytes of unauthenticated data, and 12 bytes of authenticated  
1087 data.
- 1088 3. Define the standard POSIX (also known as C) locale. Each character in the string is a single-byte US-  
1089 ASCII character.
- 1090 4. First, populate the authenticated KAD buffer, converting a sub-string consisting of the last  $2*N(a)$   
1091 characters of the full string, not including the null termination byte.
- 1092 5. When the authenticated KAD is filled, next populate the unauthenticated KAD buffer, by converting  
1093 the remaining hexadecimal character pairs in the string.

#### 1094 **3.30.1.5 Example Output**

1095 The following are examples illustrating some results of this method. In the following examples, the sizes  
1096 of the KAD for LTO4 are used. Different tape formats may utilize different KAD sizes.

1097

1098 Example 1. Full combined KAD

1099  
1100 This LTO4 tape's combined KAD contains the following data (represented in hexadecimal). For LTO4, the  
1101 unauthenticated KAD contains 32 bytes, and the authenticated KAD contains 12 bytes.  
1102

1103 Example 1a. Hexadecimal numeric data from a tape's KAD.

1104 Shaded data is authenticated by the tape drive.

1105  
1106 02 04 17 11 39 43 42 36 30 41 33 34 39 31 44 33

1107 41 41 43 36 32 42 07 F6 54 54 32 36 30 38 4C 34

1108 30 30 30 39 30 35 32 38 30 34 31 32

1109  
1110 The algorithm converts the numeric KAD data to the following 89 character null-terminated string for  
1111 storage in the Application Data field of a KMIP object's Application Specific Information attribute. The ASI  
1112 Application Name Space contains "LIBRARY-LTO4".

1113  
1114 Example 1b. Text string from KMIP ASI Application Data.

1115 Shaded characters are derived from authenticated data. The null character is represented as  
1116 <null>

1117  
1118 0204171139434236304133343931443341414336324207F65454323630384C343030303930353  
1119 23830343132<null>

1120  
1121 Example 1c. The hexadecimal values of the 89 US-ASCII characters in string 1b, from the KMIP  
1122 ASI Application Data. Note: these values are always in the range 30h-39h, or in the range 41h-  
1123 46h, or the 0h null.

1124 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  
1125 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  
1126 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

1127  
1128 For the reverse transformation, a client would retrieve the string in 1b from the server, derive the numeric  
1129 values shown in 1a, and store them to the tape format's KAD data. First, the sub-string containing the  
1130 right-most 24 characters of the full 1b string are used to derive the 12-byte authenticated KAD. The  
1131 remaining characters are used to derive the 32-byte unauthenticated KAD.

1132  
1133 Example 2. Authenticated KAD only

1134 This LTO4 tape's KAD contains the following data (represented in hexadecimal), all 12 bytes obtained  
1135 from the authenticated KAD field. There is no unauthenticated KAD data.

1136  
1137 Example 2a. Hexadecimal numeric data from a tape's KAD.

1138 Shaded data is authenticated.

1139  
1140 17 48 33 C6 20 42 10 A7 E8 05 F8 C7

1141 The algorithm converts the numeric KAD data to the following 24 character null-terminated string, for  
1142 storage in the Application Data field of a KMIP object's Application Specific Information attribute.

1143  
1144 Example 2b. Text string from KMIP ASI Application Data.



1145 Shaded characters are derived from authenticated data. The null character is represented as  
1146 <null>  
1147

1148 174833C6204210A7E805F8C7<null>

1149  
1150 For the reverse transformation, a client would derive the numeric values in 2a, and store them to the tape  
1151 format's KAD data. The right-most 24 characters of the string in 2b are used to derive the 12 byte  
1152 authenticated KAD. In this example, there is no unauthenticated KAD data.

1153  
1154 Example 3. Partially filled authenticated KAD originating from a non-KMIP method

1155 This LTO4 tape's KAD contains the following data (represented in hexadecimal). The unauthenticated  
1156 KAD contains 10 bytes, and the authenticated KAD contains 8 bytes.

1157  
1158 Since the authenticated KAD was not filled, but the unauthenticated data was populated, the method  
1159 creating this key name is potentially not backward-compatible with the KMIP key naming method. See  
1160 backward-compatibility assessment, below.

1161  
1162 Example 3a. Hexadecimal numeric data from a non-KMIP tape's KAD.

1163 Shaded data is authenticated.

1164  
1165 02 04 17 11 39 43 42 36 30 41 30 30 30 39 30 35  
1166 32 38

1167  
1168 The algorithm converts the numeric KAD data to the following 36 character null-terminated string, for  
1169 storage in the Application Data field of a KMIP object's Application Specific Information attribute.

1170  
1171 Example 3b. Text string from KMIP ASI Application Data.

1172 Shaded characters are derived from authenticated data. The null character is represented as  
1173 <null>

1174  
1175 020417113943423630413030303930353238<null>

1176  
1177 For the reverse transformation, a client would derive the same numeric values shown in 3a, and store  
1178 them to the tape's KAD. But their storage locations within the KAD now differs (see 3c). The right-most 24  
1179 characters from the text string in 3b are used to derive the 12-byte authenticated KAD. The remaining  
1180 characters are used to fill the 32-byte unauthenticated KAD.

1181  
1182 Example 3c. Hexadecimal numeric data from a tape's KAD.

1183 Shaded data is authenticated.

1184  
1185 02 04 17 11 39 43 42 36 30 41 30 30 30 39 30 35  
1186 32 38

### 1187 3.30.1.6 Backward-compatibility assessment

1188 Where all the following conditions exist, a non-KMIP solution may encounter compatibility issues during  
1189 the Read and Appended Write use cases.

- 1190 1. The tape format supports authenticated KAD, but the non-KMIP solution does not use, or only  
1191 partially uses, the authenticated KAD field.
- 1192 2. The non-KMIP solution is sensitive to data position within the combined KAD.
- 1193 3. The media was written in a KMIP environment, using this method, then moved to the non-KMIP  
1194 environment.

### 1195 3.31 Revocation Reason Codes

1196 The enumerations for the Revocation Reason attribute specified in KMIP (see table 9.1.3.2.17 in **[KMIP-  
1197 Spec]**) are aligned with the Reason Code specified in X.509 and referenced in RFC 5280 with the  
1198 following exceptions. The *certificateHold* and *removeFromCRL* reason codes have been excluded from  
1199 **[KMIP-Spec]**, since this version of KMIP does not support certificate suspension (putting a certificate  
1200 hold) or unsuspension (removing a certificate from hold). The *aaCompromise* reason code has been  
1201 excluded from **[KMIP-Spec]** since it only applies to attribute certificates, and, at this point of time, attribute  
1202 certificates are considered out-of-scope for **[KMIP-Spec]**. The *privilegeWithdrawn* reason code is  
1203 included in **[KMIP-Spec]** since it may be used for either attribute or public key certificates. In the context  
1204 of its use within KMIP it is assumed to only apply to public key certificates.

### 1205 3.32 Certificate Renewal, Update, and Re-key

1206 The process of generating a new certificate to replace an existing certificate may be referred to by  
1207 multiple terms, based upon what data within the certificate is changed when the new certificate is created.  
1208 In all situations, the new certificate includes a new serial number and new validity dates. **[KMIP-Spec]**  
1209 uses the following terminology which is aligned with the definitions found in IETF RFCs **[RFC3647]** and  
1210 **[RFC4949]**:

- 1211 • *Certificate Renewal*: The issuance of a new certificate to the subject without changing the subject  
1212 public key or other information (except the serial number and certificate validity dates) in the  
1213 certificate.
- 1214 • *Certificate Update*: The issuance of a new certificate, due to changes in the information in the  
1215 certificate other than the subject public key.
- 1216 • *Certificate Rekey*: The generation of a new key pair for the subject and the issuance of a new  
1217 certificate that certifies the new public key.

1218 The current KMIP Specification supports certificate renewals using the Re-Certify operation and certificate  
1219 updates using the Certify operation. Support for certificate rekey is not currently supported by KMIP, since  
1220 certificate rekey requires the ability to rekey an asymmetric key pair a capability not currently supported  
1221 by KMIP. Support for rekey of asymmetric key pairs, along with certificate rekey, may be considered for a  
1222 future KMIP release.

### 1223 3.33 Key Encoding

1224 Two parties receiving the same key as a Key BYTE STRING make use of the key in exactly the same  
1225 way in order to interoperate. To ensure that, it is necessary to define a correspondence between the  
1226 abstract syntax of Key and the notation in the standard algorithm description that defines how the key is  
1227 used. The next sections establish that correspondence for the algorithms AES **[FIPS197]** and Triple-DES  
1228 **[SP800-67]**.

#### 1229 3.33.1 AES Key Encoding

1230 **[FIPS197]** section 5.2, titled Key Expansion, uses the input key as an array of bytes indexed starting at 0.  
1231 The first byte of the Key becomes the key byte in AES that is labeled index 0 in **[FIPS197]** and the other  
1232 key bytes follow in index order.

1233 Proper parsing and key load of the contents of the Key for AES is determined by using the following Key  
1234 byte string to generate and match the key expansion test vectors in **[FIPS197]** Appendix A for the 128-bit  
1235 (16 byte) AES Cipher Key: 2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C.

### 1236 **3.33.2 Triple-DES Key Encoding**

1237 A Triple-DES key consists of three keys for the cryptographic engine (Key1, Key2, and Key3) that are  
1238 each 64 bits (even though only 56 are used); the three keys are also referred to as a key bundle (KEY)  
1239 **[SP800-67]**. A key bundle may employ either two or three mutually independent keys. When only two are  
1240 employed (called two-key Triple-DES), then Key1 = Key3.

1241 Each key in a Triple-DES key bundle is expanded into a key schedule according to a procedure defined in  
1242 **[SP800-67]** Appendix A. That procedure numbers the bits in the key from 1 to 64, with number 1 being  
1243 the left-most, or most significant bit. The first byte of the Key is bits 1 through 8 of Key1, with bit 1 being  
1244 the most significant bit. The second byte of the Key is bits 9 through 16 of Key1, and so forth, so that the  
1245 last byte of the KEY is bits 57 through 64 of Key3 (or Key2 for two-key Triple-DES).

1246 Proper parsing and key load of the contents of Key for Triple-DES is determined by using the following  
1247 Key byte string to generate and match the key expansion test vectors in **[SP800-67]** Appendix B for the  
1248 key bundle:

1249 Key1 = 0123456789ABCDEF

1250 Key2 = 23456789ABCDEF01

1251 Key3 = 456789ABCDEF0123

### 1252 **3.34 Using the Same Asymmetric Key Pair in Multiple Algorithms**

1253 There are mathematical relationships between certain asymmetric cryptographic algorithms such as the  
1254 Digital Signature Algorithm (DSA) and Diffie-Hellman (DH) and their elliptic curve equivalents ECDSA and  
1255 ECDH that allow the same asymmetric key pair to be used in both algorithms. In addition, one will notice  
1256 overlaps in the key format used to represent the asymmetric key pair for each algorithm type.

1257 Even though a single key pair may be used in multiple algorithms, the KMIP Specification has chosen to  
1258 specify separate key formats for representing the asymmetric key pair for use in each algorithm. This  
1259 approach keeps KMIP in line with the reference standards (e.g., NIST FIPS 186-3 **[FIPS186-3]**, ANSI  
1260 X9.42 **[X9.42]**, etc) from which the key formats for DSA, DH, ECDSA, etc. are obtained and the best  
1261 practice documents (e.g., NIST SP800-57 part 1 **[SP800-57-1]**, NIST SP800-56A **[SP800-56A]**, etc)  
1262 which recommend that a key pair only be used for one purpose.

1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308

---

## 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.

- 1309 • Attribute mutation. The possibility for the server to use attribute values different than requested by  
1310 the client if these values are not suitable for the server, and return these values in the response,  
1311 instead of failing the request.
- 1312 • Cryptographic Domain Parameters. KMIP allows a limited number of parameters to be specified  
1313 during a Create Key Pair operation. Additional parameters may be considered for the future.
- 1314 • Re-key support for other cryptographic objects. The Re-key operation is currently restricted to  
1315 symmetric keys. Applying Re-key to other cryptographic objects, such as asymmetric keys and  
1316 certificates, may be considered for the future.
- 1317 • Certificate Suspension/Unsuspendation. KMIP does not currently support certificate suspension  
1318 (putting a certificate on hold) or unsuspendation (removing a certificate from hold). Adding support  
1319 for certificate suspension/unsuspendation into KMIP may be considered for the future.
- 1320 • Namespace registration. Establishing a registry for namespaces may be considered for the  
1321 future.
- 1322 • Registering extensions to KMIP enumerations. Establishing a registry for extensions to defined  
1323 KMIP enumerations, such as in support of profiles specific to IEEE P1619.3 or other  
1324 organizations, may be considered for the future.

1325 In addition to the functionality listed above, the KMIP TC is interested in establishing a C&A (certification  
1326 and accreditation) process for independent validation of claims of KMIP conformance. Defining and  
1327 establishing this process is a candidate for work by the KMIP TC after V1.0.

---

1328 **5 Implementation Conformance**

1329 This document is intended to be informational only and as such has no conformance clauses. The  
1330 conformance requirements for the KMIP Specification can be found in the "KMIP Specification" document  
1331 itself, at the URL noted on the cover page of this document.



---

1332

## A. Acronyms

1333 The following abbreviations and acronyms are used in this document:

- 1334 3DES - Triple Data Encryption Standard specified in ANSI X9.52
- 1335 AES - Advanced Encryption Standard specified in FIPS 197
- 1336 ANSI - American National Standards Institute
- 1337 ARQC - Authorization Request Cryptogram
- 1338 ASCII - American Standard Code for Information Interchange
- 1339 CA - Certification Authority
- 1340 CBC - Cipher Block Chaining specified in NIST SP 800-38A
- 1341 CMC - Certificate Management Messages over CMS specified in RFC 5275
- 1342 CMP - Certificate Management Protocol specified in RFC 4210
- 1343 CRL - Certificate Revocation List specified in RFC 5280
- 1344 CRMF - Certificate Request Message Format specified in RFC 4211
- 1345 CVC - Card Verification Code
- 1346 DES - Data Encryption Standard specified in FIPS 46-3
- 1347 DEK - Data Encryption Key
- 1348 DH - Diffie-Hellman specified in ANSI X9.42
- 1349 FIPS - Federal Information Processing Standard
- 1350 GCM - Galois/Counter Mode specified in NIST SP 800-38D
- 1351 HMAC - Keyed-Hash Message Authentication Code specified in FIPS 198-1
- 1352 HSM - Hardware Security Module
- 1353 HTTP - Hyper Text Transfer Protocol
- 1354 HTTP(S) - Hyper Text Transfer Protocol (Secure socket)
- 1355 ID - Identification
- 1356 IP - Internet Protocol
- 1357 IPSec - Internet Protocol Security
- 1358 JKS - Java Key Store
- 1359 KEK - Key Encryption Key
- 1360 KMIP - Key Management Interoperability Protocol
- 1361 LTO4 - Linear Tape-Open 4
- 1362 MAC - Message Authentication Code
- 1363 MD5 - Message Digest 5 Algorithm specified in RFC 1321
- 1364 MGF - Mask Generation Function
- 1365 NIST - National Institute of Standards and Technology
- 1366 OAEP - Optimal Asymmetric Encryption Padding specified in PKCS#1
- 1367 PEM - Privacy Enhanced Mail specified in RFC 1421

- 1368 PGP - Pretty Good Privacy specified in RFC 1991
- 1369 PKCS - Public-Key Cryptography Standards
- 1370 POP - Proof of Possession
- 1371 POSIX - Portable Operating System Interface
- 1372 PSS - Probabilistic Signature Scheme specified in PKCS#1
- 1373 RACF - Remote Access Control Facility
- 1374 RSA - Rivest, Shamir, Adelman (an algorithm)
- 1375 SHA - Secure Hash Algorithm specified in FIPS 180-2
- 1376 SP - Special Publication
- 1377 S/MIME - Secure/Multipurpose Internet Mail Extensions
- 1378 TCP - Transport Control Protocol
- 1379 TLS - Transport Layer Security
- 1380 TTLV - Tag, Type, Length, Value
- 1381 URI - Uniform Resource Identifier
- 1382 X.509 - Public Key Certificate specified in RFC 5280
- 1383 XML - Extensible Markup Language

1384

---

## B. Acknowledgements

1385 The following individuals have participated in the creation of this specification and are gratefully  
1386 acknowledged:

1387 **Original Authors of the initial contribution:**

1388 David Babcock, HP  
1389 Steven Bade, IBM  
1390 Paolo Bezoari, NetApp  
1391 Mathias Björkqvist, IBM  
1392 Bruce Brinson, EMC  
1393 Christian Cachin, IBM  
1394 Tony Crossman, Thales/nCipher  
1395 Stan Feather, HP  
1396 Indra Fitzgerald, HP  
1397 Judy Furlong, EMC  
1398 Jon Geater, Thales/nCipher  
1399 Bob Griffin, EMC  
1400 Robert Haas, IBM  
1401 Timothy Hahn, IBM  
1402 Jack Harwood, EMC  
1403 Walt Hubis, LSI  
1404 Glen Jaquette, IBM  
1405 Jeff Kravitz, IBM  
1406 Michael McIntosh, IBM  
1407 Brian Metzger, HP  
1408 Anthony Nadalin, IBM  
1409 Elaine Palmer, IBM  
1410 Joe Pato, HP  
1411 René Pawlitzek, IBM  
1412 Subhash Sankuratipati, NetApp  
1413 Mark Schiller, HP  
1414 Martin Skagen, Brocade  
1415 Marcus Streets, Thales/nCipher  
1416 John Tattan, EMC  
1417 Karla Thomas, Brocade  
1418 Marko Vukolić, IBM  
1419 Steve Wierenga, HP

1420 **Participants:**

1421 Mike Allen, PGP Corporation  
1422 Gordon Arnold, IBM  
1423 Todd Arnold, IBM  
1424 Matthew Ball, Oracle Corporation  
1425 Elaine Barker, NIST  
1426 Peter Bartok, Venafi, Inc.  
1427 Mathias Björkqvist, IBM  
1428 Kevin Bocek, Thales e-Security  
1429 Kelley Burgin, National Security Agency  
1430 Jon Callas, PGP Corporation  
1431 Tom Clifford, Symantec Corp.  
1432 Graydon Dodson, Lexmark International Inc.  
1433 Chris Dunn, SafeNet, Inc.  
1434 Paul Earsy, SafeNet, Inc.  
1435 Stan Feather, Hewlett-Packard  
1436 Indra Fitzgerald, Hewlett-Packard

1437 Alan Frindell, SafeNet, Inc.  
1438 Judith Furlong, EMC Corporation  
1439 Jonathan Geater, Thales e-Security  
1440 Robert Griffin, EMC Corporation  
1441 Robert Haas, IBM  
1442 Thomas Hardjono, M.I.T.  
1443 Kurt Heberlein, 3PAR, Inc.  
1444 Marc Hocking, BeCrypt Ltd.  
1445 Larry Hofer, Emulex Corporation  
1446 Brandon Hoff, Emulex Corporation  
1447 Walt Hubis, LSI Corporation  
1448 Wyllys Ingersoll, Oracle Corporation  
1449 Jay Jacobs, Target Corporation  
1450 Glen Jaquette, IBM  
1451 Scott Kipp, Brocade Communications Systems, Inc.  
1452 David Lawson, Emulex Corporation  
1453 Hal Lockhart, Oracle Corporation  
1454 Robert Lockhart, Thales e-Security  
1455 Shyam Mankala, EMC Corporation  
1456 Upendra Mardikar, PayPal Inc.  
1457 Marc Massar, Individual  
1458 Don McAlister, Associate  
1459 Hyrum Mills, Mitre Corporation  
1460 Bob Nixon, Emulex Corporation  
1461 Landon Curt Noll, Cisco Systems, Inc.  
1462 René Pawlitzek, IBM  
1463 Rob Philpott, EMC Corporation  
1464 Scott Rea, Individual  
1465 Bruce Rich, IBM  
1466 Scott Rotondo, Oracle Corporation  
1467 Saikat Saha, Vormetric, Inc.  
1468 Anil Saldhana, Red Hat  
1469 Subhash Sankuratipati, NetApp  
1470 Mark Schiller, Hewlett-Packard  
1471 Jitendra Singh, Brocade Communications Systems, Inc.  
1472 Serves Singh, EMC Corporation  
1473 Terence Spies, Voltage Security  
1474 Sandy Stewart, Oracle Corporation  
1475 Marcus Streets, Thales e-Security  
1476 Brett Thompson, SafeNet, Inc.  
1477 Benjamin Tomhave, Individual  
1478 Sean Turner, IECA, Inc.  
1479 Paul Turner, Venafi, Inc.  
1480 Marko Vukolić, IBM  
1481 Rod Wideman, Quantum Corporation  
1482 Steven Wierenga, Hewlett-Packard  
1483 Peter Yee, EMC Corporation  
1484 Krishna Yellepeddy, IBM  
1485 Peter Zelechowski, Election Systems & Software  
1486 Grace Zhang, Skyworth TTG Holdings Limited

## 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.

