

1 **IEEE P1619.3™/D6**
2 **Draft Standard for Key Management**
3 **Infrastructure for Cryptographic**
4 **Protection of Stored Data**

5 Prepared by the Security in Storage Working Group of the
6 Computer Society Information Assurance Committee

7 Copyright © 2009 by the Institute of Electrical and Electronics Engineers, Inc.
8 Three Park Avenue
9 New York, New York 10016-5997, USA

10 All rights reserved.

11 This document is an unapproved draft of a proposed IEEE Standard. As such, this document is subject to
12 change. **USE AT YOUR OWN RISK!** Because this is an unapproved draft, this document must not be
13 utilized for any conformance/compliance purposes. Permission is hereby granted for IEEE Standards
14 Committee participants to reproduce this document for purposes of international standardization
15 consideration. Prior to adoption of this document, in whole or in part, by another standards development
16 organization, permission must first be obtained from the IEEE Standards Activities Department
17 (stds.ipr@ieee.org). Other entities seeking permission to reproduce this document, in whole or in part, must
18 also obtain permission from the IEEE Standards Activities Department.

19 IEEE Standards Activities Department
20 445 Hoes Lane
21 Piscataway, NJ 08854, USA

22

1 **Abstract:** <Select this text and type or paste Abstract—contents of the Scope may be used>

2 **Keywords:** <Select this text and type or paste keywords>

3

4

5

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2008 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 09 September 2008. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Incorporated.

PDF: ISBN 978-0-XXXX-XXXX-X STDXXXX
Print: ISBN 978-0-XXXX-XXXX-X STDPDXXXX

No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

1 This page is left blank intentionally.

1 Introduction

2 *[Content TBD]*

3 Notice to users

4 Laws and regulations

5 Users of these documents should consult all applicable laws and regulations. Compliance with the
6 provisions of this standard does not imply compliance to any applicable regulatory requirements.
7 Implementers of the standard are responsible for observing or referring to the applicable regulatory
8 requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in
9 compliance with applicable laws, and these documents may not be construed as doing so.

10 Copyrights

11 This document is copyrighted by the IEEE. It is made available for a wide variety of both public and
12 private uses. These include both use, by reference, in laws and regulations, and use in private self-
13 regulation, standardization, and the promotion of engineering practices and methods. By making this
14 document available for use and adoption by public authorities and private users, the IEEE does not waive
15 any rights in copyright to this document.

16 Updating of IEEE documents

17 Users of IEEE standards should be aware that these documents may be superseded at any time by the
18 issuance of new editions or may be amended from time to time through the issuance of amendments,
19 corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the
20 document together with any amendments, corrigenda, or errata then in effect. In order to determine whether
21 a given document is the current edition and whether it has been amended through the issuance of
22 amendments, corrigenda, or errata, visit the IEEE Standards Association web site at
23 <http://ieeexplore.ieee.org/xpl/standards.jsp>, or contact the IEEE at the address listed previously.

24 For more information about the IEEE Standards Association or the IEEE standards development process,
25 visit the IEEE-SA web site at <http://standards.ieee.org>.

26 Errata

27 Errata, if any, for this and all other standards can be accessed at the following URL:
28 <http://standards.ieee.org/reading/ieee/updates/errata/index.html>. Users are encouraged to check this URL
29 for errata periodically.

30

31 Interpretations

32 Current interpretations can be accessed at the following URL: [http://standards.ieee.org/reading/ieee/interp/](http://standards.ieee.org/reading/ieee/interp/index.html)
33 [index.html](http://standards.ieee.org/reading/ieee/interp/index.html).

1 **Patents**

2 Attention is called to the possibility that implementation of this standard may require use of subject matter
3 covered by patent rights. By publication of this standard, no position is taken with respect to the existence
4 or validity of any patent rights in connection therewith. The IEEE is not responsible for identifying
5 Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity
6 or scope of Patents Claims or determining whether any licensing terms or conditions provided in
7 connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable
8 or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any
9 patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further
10 information may be obtained from the IEEE Standards Association.

11 **Participants**

12 At the time this draft standard was completed, the Security in Storage Working Group had the following
13 officers:

14 **Matthew V. Ball**, *Chair*

15 **Eric Hibbard**, *Vice Chair*

16 At the time this draft standard was completed, the Security in Storage Working Group had the following
17 Membership:

18 **Robert A. (Bob) Lockhart**, *Technical Editor*

- | | | | |
|----|--------------|----|--------------|
| 19 | | | |
| 20 | Participant1 | 23 | Participant4 |
| 21 | Participant2 | 24 | Participant5 |
| 22 | Participant3 | 25 | Participant6 |
| 26 | | 26 | Participant7 |
| | | 27 | Participant8 |
| | | 28 | Participant9 |
| 29 | | | |

30 The following members of the **[individual/entity]** balloting committee voted on this standard. Balloters
31 may have voted for approval, disapproval, or abstention.

32 (to be supplied by IEEE)

33

1	CONTENTS	
2	1. Overview	2
3	1.1 Scope	2
4	1.2 Purpose	2
5	2. Normative references.....	2
6	3. Keywords, Definitions, acronyms and abbreviations	2
7	3.1 Keywords.....	2
8	3.2 Definitions	3
9	3.3 Acronyms and abbreviations	6
10	4. General Concepts and Models	7
11	4.1 Overview	7
12	4.2 Key Management Architecture Model	8
13	4.3 Key Management Conceptual Model	10
14	4.4 Key Lifecycle Model	11
15	4.5 Key Management Sequence Models.....	17
16	4.6 Key Management Operations Model.....	17
17	4.7 Key Management Object Model.....	17
18	5. Key Management Namespace	17
19	5.1 Globally unique identifier for Key Management Services	17
20	6. Key Management Objects	24
21	6.1 Key	25
22	6.2 Key Blob.....	26
23	6.3 Key_Template	26
24	6.4 ENDPOINT_TYPE	27
25	6.5 REALM (optional)	27
26	6.6 CU (Cryptographic Unit).....	28
27	6.7 KM Client	29
28	6.8 Capability	29
29	6.9 Data Sets.....	30
30	6.10 Client Groups.....	30
31	6.11 Key Groups.....	30
32	7. Key Management Policies	31
33	7.1 Key Assignment Policy	31
34	7.2 Retention Policy	32
35	7.3 Wrapping Policy	32
36	7.4 Audit Policy.....	33
37	7.5 Access/Distribution Policy	34
38	7.6 Caching Policy.....	35
39	8. Key Management Operations	35
40	8.1 Register KM Client.....	35
41	8.2 Authenticate.....	36
42	8.3 Capability Negotiation.....	36
43	8.4 Get Server Capabilities	37
44	8.5 Create/Generate Key.....	38
45	8.6 Store Key	38
46	8.7 Get Key.....	38
47	8.8 Push Audit Message	39

1 8.9 Get Random Bytes 39

2 8.10 GetStatus -- KM Client Service (optional) -- [Server initiated]..... 39

3 8.11 UpdatePending --KM Client Service (optional) [Server initiated] 40

4 8.12 GetUpdateList -- KM Server Service [Client initiated] 40

5 9. Key Management Messaging 40

6 9.1 SOAP based protocol..... 40

7 9.2 Binary command based protocol 54

8 10. Key Management Transport 55

9 Annex A (normative) Minimum Requirements to Meet P1619.3 56

10 A.1 KM Client Requirements 56

11 A.2 KM Server Requirements 56

12 Annex B (informative) Bibliography..... 58

13 Annex C (informative) Example Use Cases 59

14 C.1 Tape libraries with encrypting drives..... 59

15 C.2 Backup applications 60

16 C.3 Laptop/desktop encryption..... 60

17 C.4 NAS filer encryption..... 60

18 Annex D (informative) XML Schema Definitions 61

19 Annex E (informative) Comparison of P1619.3 Key Lifecycle Model with Other Standards 71

20 E.1 Key State Comparisons 71

21 E.2 Key Transition Comparisons..... 71

22 E.3 Key Time Period Comparisons 72

23 Annex F (informative) Discussion of SO_GUID formats 75

24 F.1 Comparison Chart of Name Spaces..... 75

25 F.2 SO_Family Methods..... 76

26 F.3 SO_Family Advantages, Potential Concerns and Solutions 81

27

28

1 **Draft Standard for Key Management**
2 **Infrastructure for Cryptographic**
3 **Protection of Stored Data**

4

5

1 1. Overview

2 1.1 Scope

3 This standard specifies an architecture for the key management infrastructure for cryptographic protection
4 of stored data, describing interfaces, methods and algorithms.

5 1.2 Purpose

6 This standard defines methods for the storage, management, and distribution of cryptographic keys used for
7 the protection of stored data. This standard augments existing key management methodologies to address
8 issues specific to cryptographic protection of stored data. This includes stored data protected by compliant
9 implementations of other standards in the IEEE 1619 family.

10 2. Normative references

11 The following referenced documents are indispensable for the application of this document. For dated
12 references, only the edition cited applies. For undated references, the latest edition of the referenced
13 document (including any amendments or corrigenda) applies.

- 14 — NIST Special Publication 800-57 March 2007, Recommendations for Key Management
- 15 — ISO/IEC 11770-1:1996 Information technology - Security techniques - Key management - Part 1:
16 Framework
- 17 — SOAP v 1.1 (<http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>)
- 18 — IEEE Std 1003.1-2001 Portable Operating System Interface for Computer Environments (POSIX)
- 19 — IEEE Std 1363-2000 Standard Specifications For Public-Key Cryptography
- 20 — IEEE Std 1619-2007 Standard for Cryptographic Protection of Data on Block-Oriented Storage
21 Devices
- 22 — IEEE Std 1619.1-2007 Standard for Authenticated Encryption with Length Expansion for Storage
23 Devices
- 24 — RFC1034 Domain Names – Concepts and Facilities, P. Mockapetris
- 25 — RFC3548 The Base16, Base32, and Base64 Data Encodings, S. Josefsson, Ed.
- 26 — RFC3986 Uniform Resource Identifier (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L.
27 Masinter
- 28 — RFC4234 Augmented BNF for Syntax Specifications, D. Crocker Editor & P. Overell

29 3. Keywords, Definitions, acronyms and abbreviations

30 For the purposes of this document, the following terms and definitions apply. The Authoritative Dictionary
31 of IEEE Standard Terms [B8] should be referenced for terms not defined in this clause.

32 3.1 Keywords

33 For the purposes of this standard, the following terms are keywords.

34 **can:** A keyword indicating a capability (can equals is able to).

35 **required:** See shall.

1 **may:** A keyword indicating a course of action permissible within the limits of this standard (may equals is
2 permitted to).

3 **must:** A keyword used only to describe an unavoidable situation that does not constitute a requirement for
4 compliance to this standard.

5 **shall:** A keyword indicating a mandatory requirement strictly to be followed in order to conform to this
6 standard and from which no deviation is permitted.(shall equals is required to).

7 **shall not:** A phrase indicating an absolute prohibition of this standard.

8 **should:** A keyword indicating that among several possibilities one is recommended as particularly suitable,
9 without mentioning or excluding others; or that a certain course of action is preferred but not necessarily
10 required or that (in the negative form) a certain course of action is deprecated but not prohibited (should
11 equals is recommended to).

12 **3.2 Definitions**

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

15 **3.2.1 Access control:** Restricts access to resources only to privileged entities.

16 **3.2.2 Association:** A relationship for a particular purpose. For example, a key is associated with the
17 application or process for which it will be used.

18 **3.2.3 Authentication:** The act of verifying the identity of an entity

19 **3.2.4 Availability:** Timely, reliable access to information.

20 **3.2.5 Compromise:** The unauthorized disclosure, modification, substitution, or use of data (e.g., keying
21 material and other security related information).

22 **3.2.6 Confidentiality:** The property that sensitive information possesses if it is not to be disclosed to
23 unauthorized entities. In a general information security context: preserving authorized restrictions on
24 information access and disclosure, including means for preserving personal privacy and proprietary
25 information.

26 **3.2.7 Control plane:** Plane where requirements of encryption/decryption are determined for data traversing
27 or stored in the data plane.

28 **3.2.8 Control plane operations:** In the context of encryption of stored data, control plane operations
29 include those operations necessary to identify the encryption requirements of the stored data, generate
30 cryptographic information necessary to properly configure the Cryptographic Units, retrieve this
31 cryptographic information, and configure the Cryptographic Units. In addition, control plane operations
32 include all operations necessary to protect and distribute this cryptographic information throughout a key
33 management infrastructure. Control plane operations tend to be lower performance operations that are
34 decoupled from the data plane where the actual encryption of stored data takes place.

35 **3.2.9 Cryptographic key:** A parameter used in conjunction with a cryptographic algorithm that determines
36 its operation in such a way that an entity with knowledge of the key can reproduce or reverse the operation,
37 while an entity without knowledge of the key cannot. Examples include:

38 the transformation of plaintext data into ciphertext data,
39 the transformation of ciphertext data into plaintext data,
40 the computation of a digital signature from data,
41 the verification of a digital signature,
42 the computation of an authentication code from data,
43 the computation of a shared secret that is used to derive keying material.

1 **3.2.10 Cryptographic material:** Any encrypted data, keys, policies or other information that is used in
2 cryptographic operations or stored in encrypted format.

3 **3.2.11 Cryptographic unit:** An entity capable of using cryptographic algorithms to encrypt data being
4 written to and decrypt data read from storage media. A Cryptographic Unit lies in both the data plane
5 (where encryption and decryption take place) and the control plane (where data set information,
6 cryptographic keys, and security parameters are transported between itself and the key management client).

7 **3.2.12 Data plane:** Region where data traverses, is stored or has operations performed on it

8 **3.2.13 Data plane operations:** In the context of encryption of stored data, data plane operations occur
9 when encryption users write plaintext data through the cryptographic unit where the data is encrypted and
10 deposited as ciphertext to the storage media. Likewise, data plane operations also occur when encryption
11 users read plaintext data through the cryptographic unit that decrypts ciphertext data retrieved from the
12 storage media. Data plane operations tend to be high performance operations that are decoupled from the
13 control plane where key management operations typically take place.

14 **3.2.14 Data set:** A collection of data, independent of structure, media or transport that is encrypted using a
15 common key.

16 **3.2.15 Data set attribute:** A specific attribute (e.g. file path, tape volume id, server IP address, a range of
17 disk blocks or identifier) that can be used to define a policy for how one or more data sets are to be
18 protected.

19 **3.2.16 Data set binding:** A part of the Key Exchange Structure, in which a specific data set is identified,
20 together with a specific key, signifying that the data that meets the matching rules was encrypted by the
21 given key.

22 **3.2.17 Encoded octet:** A hexadecimal tri-graph where the value of the octet is represented in the two
23 base16 encoded characters that follow the '%' character.

24 **3.2.18 Encryption entity:** A software and/or hardware entity that is capable of accepting cryptographic
25 keys and using them to encrypt and decrypt data that is written and read, respectively, to persistent data
26 storage media. Encryption entities are composed of a Cryptographic Unit and a key management client.

27 **3.2.19 Encryption users:** Defines all users, servers, applications, and systems that require the stored data
28 media used for their persistent storage requirements be protected by encryption.

29 **3.2.20 Hexadecimal tri-graph:** A '%' character followed by two characters from the base16 encoding
30 character set as defined by RFC 3548. The base16 encoding character set consists of the following 16
31 characters:

32 0 1 2 3 4 5 6 7 8 9 A B C D E F

33 Note that lower case letters are not part of the hexadecimal encoding character set.

34 For purposes of this document each character is presented and stored as a full octet and for length
35 calculation, a base16 tri-graph shall count as three octets.

36 **3.2.21 Key attributes:** Additional information about a key that includes the key identifier, key contents,
37 time stamps, and other associated information.

38 **3.2.22 Key blob:** A consolidated set of data containing the cryptographic key with or without additional
39 attributes. Differentiated by endpoint, and is constructed from an immutable key value, the storage and
40 representation of which is outside of this standard

41 **3.2.23 Key distribution:** The transport of a key and other keying material from an entity that either owns
42 the key or generates the key to another entity that is intended to use the key.

43 **3.2.24 Key exchange structure:** An encapsulation of a key, together with associated data and policies
44 governing the use of the key.

45 **3.2.25 Key group:** A static or dynamic collection of cryptographic keys.

1 NOTE— see 6.11

2 **3.2.26 Key management:** The activities involving the handling of cryptographic keys and other related
 3 security parameters during the entire life cycle of the keys, including their generation, storage,
 4 establishment, entry and output, and destruction.

5 **3.2.27 Key management API:** A collection of reference programming interfaces to the key management
 6 software library; one interface each for the C, C++, and Java language bindings.

7 **3.2.28 Key management client:** A component of an Encryption Entity that serves two primary functions:
 8 1) communicates with key management servers to acquire cryptographic keys, and 2) communicates with
 9 Cryptographic Units to load and activate these keys for their use in encrypting and decrypting data that is
 10 written and read, respectively, to data storage media.

11 **3.2.29 Key management client-server operations:** Defines the set of high level operations and their
 12 message and transport mechanisms that allow key management clients to request key management services
 13 from key management servers and, likewise, to allow key management servers to respond to these key
 14 management service requests. These operations utilize a well defined set of key management objects for the
 15 communication of key management related information between the clients and servers.

16 **3.2.30 Key management infrastructure:** The framework and services that provide for the generation,
 17 production, distribution, control, accounting, and destruction of all cryptographic material, including
 18 cryptographic keys. It includes all elements (hardware, software, other equipment, and documentation);
 19 facilities; personnel; procedures; standards; and information products that form the system that distributes,
 20 manages, and supports the delivery of cryptographic services to end users.

21 **3.2.31 Key management objects:** TBD

22 **3.2.32 Key management service:** A system that provides one or more services to provide control of
 23 cryptographic keys, policies and logging.

24 **3.2.33 Key management server:** A software and/or hardware entity that is capable of generating, storing,
 25 protecting, and distributing cryptographic keys and other security related information necessary to support
 26 the operation of Encryption Entities. In addition, they may also support security services (e.g. audit
 27 services, backup/archive services, security policy management, and access control services).

28 **3.2.34 Key management software library:** A software library developed by the P1619.3 Task Group that
 29 provides a reference implementation of a portion of the key management client that generates Key
 30 Management Client-Server Operations. The interface to the library is the Key Management API. The
 31 library uses industry standard SSL/TLS and TCP interface calls for generating Key Management Client-
 32 Server Operations over the network. The use of this library is optional, but it can be used to facilitate the
 33 rapid development of compliant key management clients by implementers.

34 **3.2.35 Key management user:** One or more entities that are responsible for the configuration and
 35 management of the key management infrastructure.

36 **3.2.36 Key object:** a data structure consisting of the key and its attributes.

37 **3.2.37 KM filename:** A filename consisting of only hexadecimal tri-graphs and characters from the POSIX
 38 portable filename character set. (see 5.1.2)

39 **3.2.38 KM FQDN:** A fully qualified domain name consisting of only hexadecimal tri-graphs and the
 40 characters allowed in a domain name as defined by RFC 1034. The RFC 1034 domain name characters
 41 are:

42 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

43 a b c d e f g h i j k l m n o p q r s t u v w x y z

44 0 1 2 3 4 5 6 7 8 9 .

45 The last character is a period (%2E).

1 In anticipation of internationalization of domain names, all characters that are not in the RFC 1034 domain
 2 name character set shall be represented as encoded octets. No other characters of the KM FQDN shall be
 3 encoded as a hexadecimal tri-graph.

4 **3.2.39 KMS symlink:** An indirect reference to a SO_GUID. When a SO_GUID resolves to a KMS
 5 symlink, the value of that KMS symlink is treated as a SO_GUID.

6 **3.2.40 Policy:** A rule that defines one or more requirements for how a key or group of keys are generated,
 7 used, distributed, retained, audited and/or stored.

8 **3.2.41 POSIX portable filename character set:** The set of characters defined by the IEEE Std 1003.1-
 9 2001 Section 3.276 portable filename character set. This set consists of the following 65 characters:

10 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

11 a b c d e f g h i j k l m n o p q r s t u v w x y z

12 0 1 2 3 4 5 6 7 8 9 . _ -

13 The last three characters are the period (%2E), underscore (%5F), and hyphen (%2D) characters
 14 respectively. Upper- and lowercase letters shall retain their unique identities (i.e., they are case sensitive).

15 **3.2.42 POSIX portable filename:** An alphanumeric octet followed by zero or more octets from the POSIX
 16 portable filename character set. Upper- and lowercase letters shall retain their unique identities (i.e., they
 17 are case sensitive).

18 **3.2.43 Record ID:** A 12 octet unique identifier for a security object within a specific SO_Domain.
 19 Currently used in the URI name space (See 5.1.2).

20 **3.2.44 Security object:** An object that contains values that affect the security of the key management
 21 service if disclosed or modified. Examples of a security object: a cryptographic key, a policy, or set of
 22 policies, a security log entry, a user or client description, or authorization.

23 **3.2.45 Security object globally unique identifier (SO_GUID):** a value that uniquely identifies a
 24 particular security object and that has a low probability of having the same value as another independent
 25 SO_GUID. See also security object.

26 **3.2.46 Storage media:** Any device that is used to store and retrieve data over extended periods of time.

27 **3.2.47 Universal resource identifier:** A compact sequence of characters that identifies an abstract or
 28 physical resource defined in IETF RFC 3986.

29 **3.3 Acronyms and abbreviations**

30 CN Client Node

31 CU Cryptographic Unit

32 FQDN Fully Qualified Domain Name

33 KPF Key Processing Facility

34 KM Key Management

35 KM API Key Management Application Programming Interface

36 KM Client Key Management Client

1	KM Server	Key Management Server
2	KM SW Lib	Key Management Software Library
3	KM User	Key Management User
4	KMCS Ops	Key Management Client-Server Operations
5	KMI	Key Management Infrastructure
6	KMS	Key Management Service
7	KMSS Ops	Key Management Server-Server Operations
8	PRNG	Pseudorandom Number Generator
9	RNG	Random Number Generator
10	SA	Service Agent
11	SO_GUID	Security Object Globally Unique Identifier
12	URI	Universal Resource Identifier

13 **4. General Concepts and Models**

14 **4.1 Overview**

15 Cryptographic mechanisms are used as a strong way to provide secure access control to stored data.
 16 Encrypting the stored data makes it accessible only to entities that have access to the key that can decrypt
 17 the data. Therefore, the proper management of cryptographic keys is essential to the effective use of
 18 cryptography for security. Mismanagement of keys may grant access to unauthorized parties or render
 19 critical information inaccessible to all.

20 Intelligent key management adds security by selectively distributing keys according to an organization's
 21 information security policy. There are several advantages of centralizing key management:

- 22 — Centralized policy management
- 23 — Centralized audit and reporting
- 24 — Segregate the security administrators from the storage or IT administrators.

25 Therefore this standard specifies a key management infrastructure in which a set of key management
 26 servers (KM Server) provide key management services to clients. The key management services defined in
 27 this standard include distribution, generation, availability, enforcement, storage, archival and audit of keys
 28 and policies.

29 A critical issue is how to assign encryption keys to broadly defined data sets. This standard proposes a
 30 matching system that allows the KM Server to provide keys to clients based on those attributes most easily
 31 available to the clients, namely data or storage identifiers.

32 The messages sent between clients and KM Server are encoded in an extensible Key Exchange Structure.
 33 This structure associates a key to a data set, and lists policies governing key use.

1 **4.1.1 Key Management Models**

2 The following sub clauses contain a number of abstract models that define the high level architecture for
3 environments and systems that provide management for encryption keys and their related security
4 attributes, parameters, and objects used for the protection of stored data.

5 Because of the complex nature of key management, several models have been developed to provide various
6 viewpoints into the architecture of a key management infrastructure and describe the entities and objects
7 that make up these environments. The key management models presented in the standard are as follows:

- 8 • Key Management Architecture Model – the basic, high level model for key management
9 infrastructures that provides a foundation for the remaining models and detailed information
10 developed in the later sections of this standard.
- 11 • Key Management Conceptual Model – provides the basic model for the internal and external
12 organization of the primary components in a key management infrastructure.
- 13 • Key Lifecycle Model – the basic state machine model that encompasses the entire set of standard
14 key states and the transitions between these states as maintained by the key management systems
15 that implement this standard.
- 16 • Key Management Sequence Models – a series of models that describe the high level interactions
17 between the primary components in the Key Management Conceptual Model.
- 18 • Key Management Object Models
- 19 • Key Management Operation Models

20 **4.2 Key Management Architecture Model**

21 The protection of stored data using encryption requires the use of a key management infrastructure where
22 the encryption keys and other security related parameters can be adequately managed and protected. This
23 standard defines a basic architecture for a key management infrastructure which includes the definitions
24 and relationships between components within this environment.

25 Figure 1 illustrates the architecture model for the key management infrastructure defined in the standard.

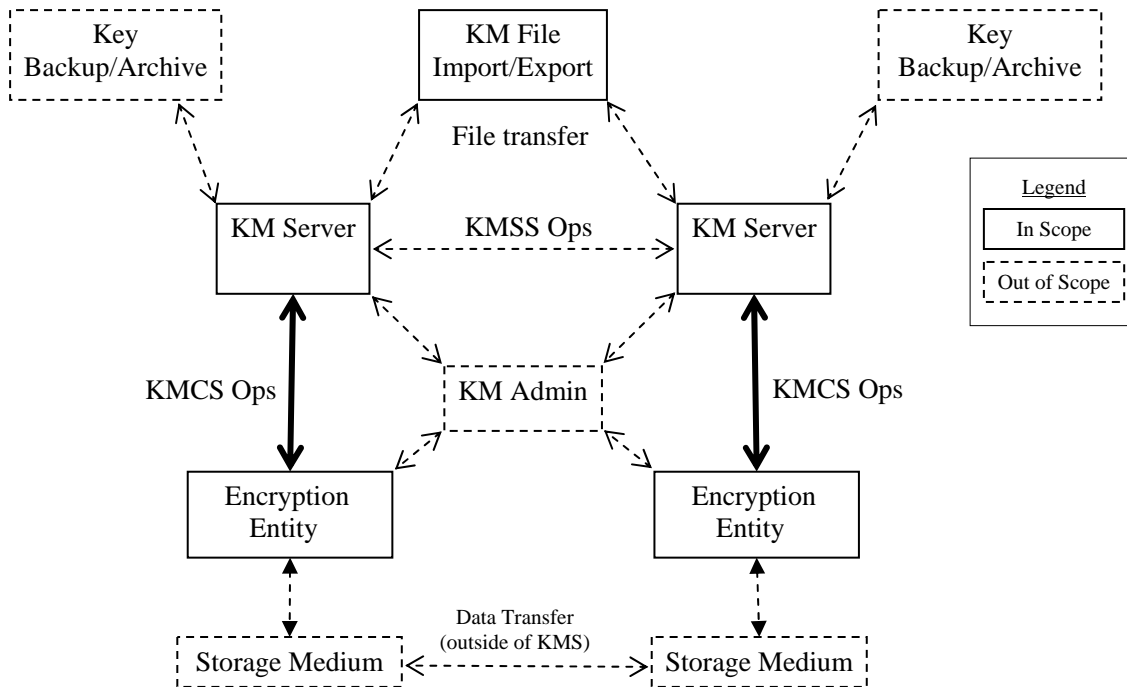


Figure 1 Key Management Architecture Model

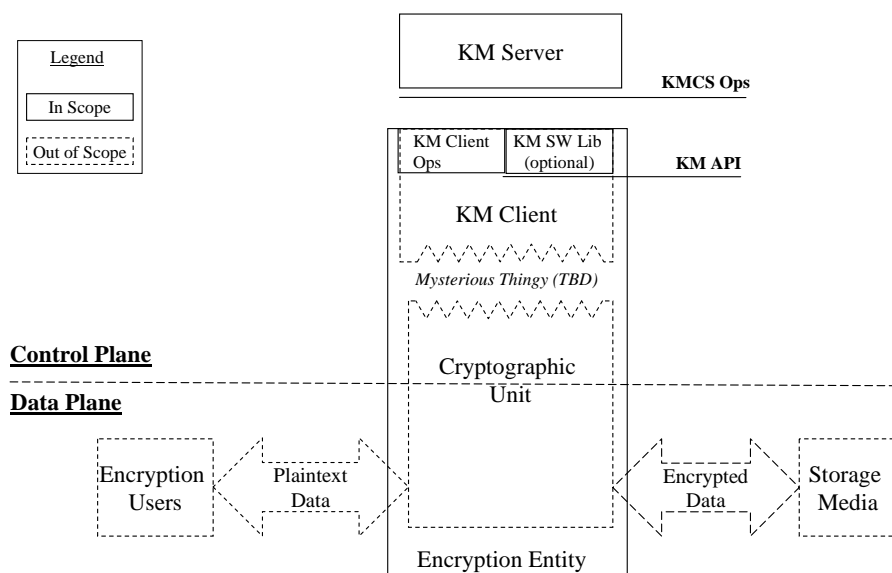
The following list describes each component within Figure 1 in more detail:

- KM Server: (in scope) The Key Management server is a software and/or hardware entity that is capable of generating, storing, protecting, and distributing cryptographic keys and other security related information necessary to support the operation of Encryption Entities. In addition, it may also support security services (e.g. audit services, backup/archive services, security policy management, and access control services).
- Encryption Entity: (in scope) An entity or collection of entities that is able to receive encryption keys, enforce policy, and encrypt or decrypt data to and from a storage medium (see 4.3).
- Storage Medium: (out of scope) This may include magnetic, optical, or solid-state memory devices (see 3.2.46).
- Key Backup and Archive: (out of scope) This function is performed by a software and/or hardware entity that provides secure, long-term storage of keys. This service may include key backup services for securing copies of currently active keying material, as well as key archive services to provide a repository for non-active key material of historical interest.
- KM File Import/Export: (in scope) The Key Management File Import and Export service is a hardware and/or software entity that provides file exchange services between KM servers.
- KM Admin: (out of scope) The key management administrator is an entity that is responsible for the configuration and management of the key management infrastructure. There may be several KM Administrators, but all must be authenticated with the management servers. The role, functions and authentication methods of the KM Admin is outside the scope of this document.
- KMSS Ops: (out of scope) The Key Management Server-Server Operations define the set of high level operations, their messages, and transport mechanisms that allow KM Servers to communicate with one another in order to facilitate the delivery of key management services that can not be accomplished with a single KM Server. Server-to-server operations facilitate advanced key management services, e.g. hierarchical key distribution and availability of key services for key management infrastructures.

- 1 — KMCS Ops: (in scope) The Key Management Client-Server Operations define the set of high level
 2 operations, their messages, and transport mechanisms that allow KM Clients to request key
 3 management services from KM Servers and to allow KM Servers to respond to these key
 4 management service requests. These operations use a set of key management objects for the
 5 communication of key management related information between the clients and servers.
 6 — In Scope: The KM Servers, KM Clients, Encryption Entities and KMCS Operations are the subject
 7 of the standard; all other entities are included for completeness, but are not specified in the
 8 standard.

9 4.3 Key Management Conceptual Model

10 Figure 2 illustrates the conceptual model for the key management infrastructure defined in this standard.



11
 12 **Figure 2 Key Management Conceptual Model**

13 The following list describes each component within Figure 2:

- 14 — Control Plane: Where data set information, cryptographic keys, and security parameters are
 15 transported between the Cryptographic Unit and the key management client
 16 — Data Plane: Where data encryption and decryption take place
 17 — KM Server: (in scope) See 4.2
 18 — Encryption Users: (out of scope) Defines all users, servers, applications, and systems that require
 19 the stored data be encrypted when stored on their storage media.
 20 — Storage Media: (out of scope) Anything capable of storing non-volatile data.
 21 — Encryption Entity: (in scope) An entity that contains the following logical or physical components:

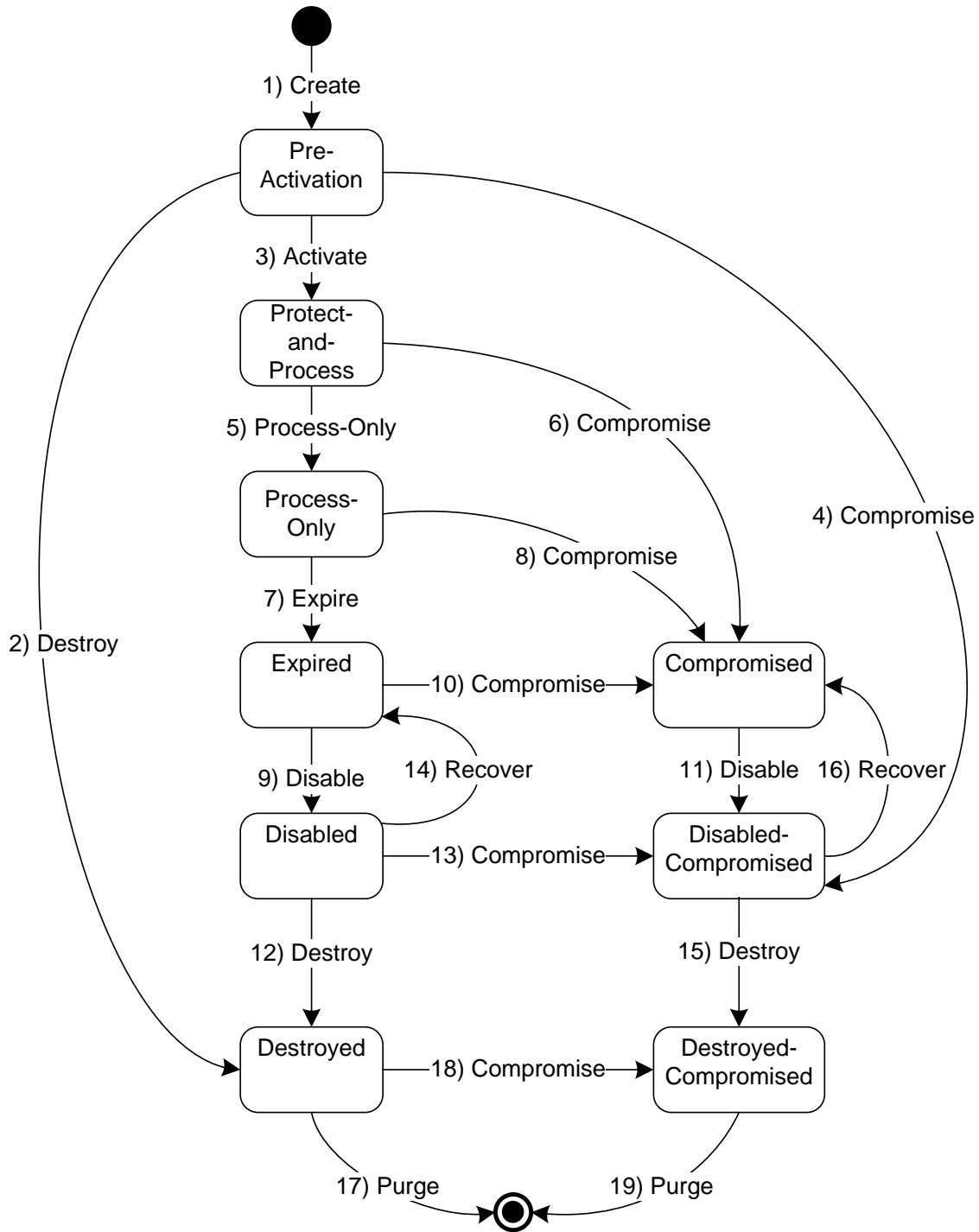
- 1 — KM Client (In Scope): A component of an Encryption Entity that serves several
 2 functions:
 3 i) Communicates with key management servers to acquire cryptographic keys;
 4 ii) Communicates with Cryptographic Units to load and activate these keys for
 5 their use in encrypting and decrypting data that is written to and read from,
 6 respectively, to data storage media, and;
 7 — The KM Client enforces these functions in one of two ways:
 8 i) KM Client Ops: (out of scope) The key management client enforces
 9 these functions through the appropriate implementation and use of the
 10 KM Client Operations.
 11 ii) KM SW Lib: (in scope) These functions may be enforced through the
 12 use of a standard Key Management Software Library as defined in the
 13 standard.
 14 — Cryptographic Unit: (out of scope) An entity capable of using cryptographic algorithms
 15 to encrypt data being written to and decrypt data read from storage media.

16 **4.4 Key Lifecycle Model**

17 A crucial aspect of this key management standard is the concise definition for the states of encryption keys
 18 and the various transitions that can occur between these states. This definition of states of transitions
 19 between states for encryption keys is commonly referred to as the key lifecycle. Several standards bodies
 20 have attempted to define general key lifecycle models with their respective key state and transition
 21 definitions. These models were typically created in the context of using keys for protecting “data in flight”
 22 over communications links. The ephemeral nature of the keys and protected communication “sessions”
 23 does not lend itself well to the using these models “as-is” in the protection of stored data, or “data at rest”
 24 as it is commonly known.

25 A modified key lifecycle model is required that adequately defines the key states and transitions between
 26 these states to ensure the highest levels of protection for stored data. The key lifecycle model defined
 27 herein is based largely on the key lifecycle models defined in NIST SP800-57 Part 1, and IEC/ISO 11770-
 28 1, but with additional states, transitions, and clarifications of these states and transitions in context of key
 29 management for the protection of stored data environments. The informative annex of this standard
 30 provides a concise comparison of the key lifecycle states and transitions presented in this standard, and
 31 compares them to the key lifecycle states and transitions presented in the key lifecycle models of applicable
 32 international and national standards.

33 Figure 3 illustrates the key lifecycle model as defined in this standard.



1
2
3
4
5
6
7
8

Figure 3 Key Lifecycle Model

The key lifecycle model consists of a set of key states in which any particular encryption key can exist in at any particular time. In addition, this model indicates the allowable transitions between states. The definitions of these transitions include the events or actions necessary to cause a key to transition from one state to the next. These events or actions include time out events or user/management controlled actions. To complete the key lifecycle model, the definition of a set of time periods must be defined that control the automated transition of keys between several of the states in the key lifecycle model.

1 4.4.1 Key State Definitions

2 The following are the formal descriptions of the key states defined in the key lifecycle model:

- 3 — Pre-Activation: The key has been generated but is not yet in use or distributed to any KM Client or
4 Cryptographic Unit. A key in this state can only exist in the KM Server. Such a key can be
5 distributed to a KM Client.
- 6 — Protect-and-Process: A key in this state can be used for both encryption (i.e., to protect information)
7 and decryption (to process information). A key is placed into this state when it is initially
8 given to a KM Client, e.g., it is activated. The activation is done when a KM Client requests a
9 new key. If a key is created by a KM Client or Cryptographic Unit and is then given to the
10 KM Server, the key shall be immediately placed into Protect-and-Process state. A key shall
11 remain in this state until the encryption period passes.
- 12 — Process-Only: A key in this state can be used for decryption but not encryption. When a KM
13 Client/Cryptographic Unit determines that none of the keys available to it (e.g., for a specific
14 tape cartridge or disk volume that is being read or written) are in the protect-and-process state,
15 and it needs to perform encryption, it should create a new key. Keys transition from Protect-
16 and-Process to Process-Only when the Encryption Period for the key expires, or as a result of
17 an administrative action. A key shall remain in the Process-Only state until the Crypto
18 Period passes. At that time the key shall be expired and move to the Expired state.
- 19 — Expired: An expired key is a key that has had its Crypto Period expire, but it may still be needed to
20 process (decrypt) information. Keys in this state may be used to process (decrypt) data only.
21 An expired key can be delivered to a KM Client, but the Cryptographic Unit should not use
22 the key to encrypt data. A key shall remain in this state until the key's Disable Period expires
23 or as a result of an administrative action. At that time the key shall be disabled and it shall
24 move to the Disabled state.
25 The difference between keys in the Process-Only and Expired states is subtle. As far as KM
26 Clients or Cryptographic Units are concerned, the states are equivalent. The difference is
27 significant only to KM Servers. A key in the Process-Only state is still in the NIST
28 operational phase and would routinely be delivered to KM Clients. A key which has had its
29 Crypto Period expire has moved into the NIST post-operational phase, and may have
30 additional restrictions imposed on its delivery to KM Clients.
- 31 — Disabled: A disabled key is a key that is still known to the KM Server but it shall not be delivered
32 to a KM Client. A disabled key's key material remains intact in the KM Server. A key shall
33 remain in the Disabled state until either the key's Destruction Period expires, or
34 administrative action places the key in the Destroyed state, Compromised state, or recovers
35 the key to the Expired state. If the key's Destruction Period expires, the key shall be destroyed
36 and shall transition to the Destroyed state.
- 37 — Compromised: Keys are compromised when they are released to or discovered by an unauthorized
38 entity. Compromised keys should not be used to protect information, but may need to be used
39 to process information. The KM Server cannot determine if a key has been compromised.
40 An administrative action is needed to inform a KM server that a key has been compromised.
41 A compromised key shall be delivered to a KM Client, but the Cryptographic Unit should not
42 use the key to encrypt data. Just as with Expired keys, a KM Server may impose additional
43 restrictions on the delivery of compromised keys to a KM Client. A key shall remain in this
44 state until the Disable Period expires. At that time the key shall be disabled and move to the
45 Disabled-Compromised state.
- 46 — Disabled-Compromised: A key that is both disabled and compromised. Disabled and compromised
47 keys shall never be delivered to KM Clients. A key shall remain in the disabled compromised
48 state until either the Destruction Period expires or the key is recovered to the Compromised
49 state by administrative action. If the Destroy Period expires, the key shall be destroyed and
50 moved to the Destroyed-Compromised state.

- 1 — Destroyed: A destroyed key is a key which has had its key material removed from the KM Server.
2 Information or metadata about the key may be retained by the KM Server. Destroyed keys
3 cannot be delivered to a KM Client. Keys can be destroyed by either an administrative action
4 in the KM Server, or by the expiration of the Destruction Period of the key while it is in the
5 Disabled or Disabled-Compromised states.
- 6 — Destroyed-Compromised: Similar to keys in the Destroyed state, but the key was compromised
7 before or after destruction.
- 8 — Terminal (Purged): Purged is the terminal state for keys. A purged key is a key that no longer
9 exists in the KM Server in any form. Neither the key material nor any metadata about the key
10 is known to the KM Server. A purged key cannot be delivered to a KM Client.

11 4.4.2 Key State Transition Definitions

12 The following are the formal descriptions of the key state transitions defined in the key lifecycle model:

- 13 — (1) Create: When a key is created by a KM Server, it begins in the pre-activation state. This
14 transition occurs as soon as a key is generated within the KM Server, such as a key being
15 generated by a random number generator (RNG).
- 16 — Transition 1 applies to newly created keys; the key immediately transitions to Pre-Activation state
17 upon its creation.
- 18 — (2) Destroy: A key in the Pre-Activation state may be moved directly to the Destroyed state. If the
19 key has never been activated and is no longer required, but there is a requirement to maintain
20 information about the key, the key may be destroyed. This transition can only occur as a result
21 of administrative action.
- 22 — Transition 2 applies to keys in the Pre-Activation state; the key transitions to the Destroyed state.
- 23 — (3) Activate: A key transitions from the Pre-Activation state to the Protect-and-Process state when
24 it is available for use. This transition occurs when the key is assigned for use by a KM Client.
25 There is a bit of confusion around the terminology, since a KM Client may view this action as
26 “creating a key” even though the KM Server views this as assigning a previously generated
27 key.
- 28 — Transition 3 applies to keys in the Pre-Activation state; the key transitions to the Protect-and-
29 Process state.
- 30 — (4) Compromise: A key transitions to the Compromised state when it has been determined to have
31 been released to or discovered by an unauthorized entity. The KM Server cannot determine
32 that this has happened, so this transition occurs as the result of an administrative action.
- 33 — Transition 4 applies to keys in the Pre-Activation state; the key transitions to the Compromised
34 state.
- 35 — (5) Process-Only: A key transitions from the Protect-and-Process state to the Process-Only state
36 when a period of time equal the Encryption Period has expired after the key is activated. This
37 transition may also occur as a result of administrative action. Some KM Clients and
38 Cryptographic Units may be unable to strictly enforce this transition.
- 39 — Transition 5 applies to keys in the Protect-and-Process state; the key transitions to the Process-Only
40 state.
- 41 — (6) Compromise: Same actions as for the (4) Compromise transition.
- 42 — Transition 6 applies to keys in the Protect-and-Process state; the key transitions to the
43 Compromised state.
- 44 — (7) Expire: A key transitions from active to Expired when its Crypto Period expires. This
45 transition may also occur as a result of administrative action.
- 46 — Transition 7 applies to keys in the Process-Only state; the key transitions to the Expired state.
- 47 — (8) Compromise: Same actions as for the (4) Compromise transition.

- 1 — Transition 8 applies to keys in the Process-Only state; the key transitions to the Compromised state.
- 2 — (9) Disable: This transition shall occur after the disable period for a key passes. This transition can
3 also occur as a result of administrative actions. The key material and its metadata shall be
4 retained by the KM Server. The key shall no longer delivered to KM Clients.
- 5 — Transition 9 applies to keys in the Expired state; the key shall transition to the Disabled state.
- 6 — (10) Compromise: Same actions as for the (4) Compromise transition.
- 7 — Transition 10 applies to keys in the Expired state; the key transitions to the Compromised state.
- 8 — (11) Disable: Same actions as for the (9) Disable transition.
- 9 — Transition 11 applies to keys in the Compromised state; the key transitions to the Disabled-
10 Compromised state.
- 11 — (12) Destroy: A key can be destroyed when it is no longer needed. When the Destruction Period
12 for a key expires, it shall be destroyed. This transition can also occur as a result of
13 administrative action. Active keys (keys in the Protect-and-Process or Protect-Only states)
14 cannot be destroyed; only keys that have been previously activated and are now in the
15 Disabled or Disabled-Compromised states can be destroyed. When a key is destroyed, the
16 key material is removed from the KM Server. Metadata about the key is retained in the KM
17 Server.
- 18 — Transition 12 applies to keys in the Disabled state; the key transitions to Destroyed state.
- 19 — (13) Compromise: Same actions as for the (4) Compromise transition.
- 20 — Transition 13 applies to keys in the Disabled state; the key transitions to Disabled-Compromised
21 state.
- 22 — (14) Recover: When a disabled key is required to be delivered to KM Clients, it can be recovered.
23 This occurs as a result of an administrative action.
- 24 — Transition 14 applies to key in the Disabled state; the key transitions to Expired state.
- 25 — (15) Destroy: Same actions as for the (12) Destroy transition.
- 26 — Transition 15 applies to key in the Disabled-Compromised state; the key transitions to the
27 Destroyed-Compromised state.
- 28 — (16) Recover: Same actions as for the (14) Recover transition.
- 29 — Transition 16 applies to keys in the Disabled-Compromised state; the key transitions to the
30 Compromised state.
- 31 — (17) Purge: The action of purging a key removes all information about the key from the KM
32 Server's key records. Neither the key material nor the key value shall be retained by the KM
33 Server. Note, however, that audit logs or other indirect information in the KM Server may
34 still contain information about the key.
- 35 — Transition 17 applies to keys in the Destroyed state; the key transitions to the Purged state.
36 However, since the key does not exist in the KM Server, there is no record for the key
37 showing the Purged state.
- 38 — (18) Compromise: Same actions as for the (4) Compromise transition.
- 39 — Transition 18 applies to keys in the Destroyed state; the key transitions to the Destroyed-
40 Compromised state.
- 41 — (19) Purge: Same actions as for the (17) Purge transition.
- 42 — Transition 19 applies to keys in the Destroyed-Compromised state; the key transitions to the Purged
43 state. However, since the key does not exist in the KM Server, there is no record for the key
44 showing the purged state.

1 **4.4.3 Key Time Period Definitions**

2 Many of the key states and transitions defined in the key lifecycle model are dependent on one or more
 3 time related attributes associated with the keys. These time related attributes are typically categorized as
 4 timer values or timeout periods that are activated when keys enter certain states and their expiration cause
 5 transitions to certain states.

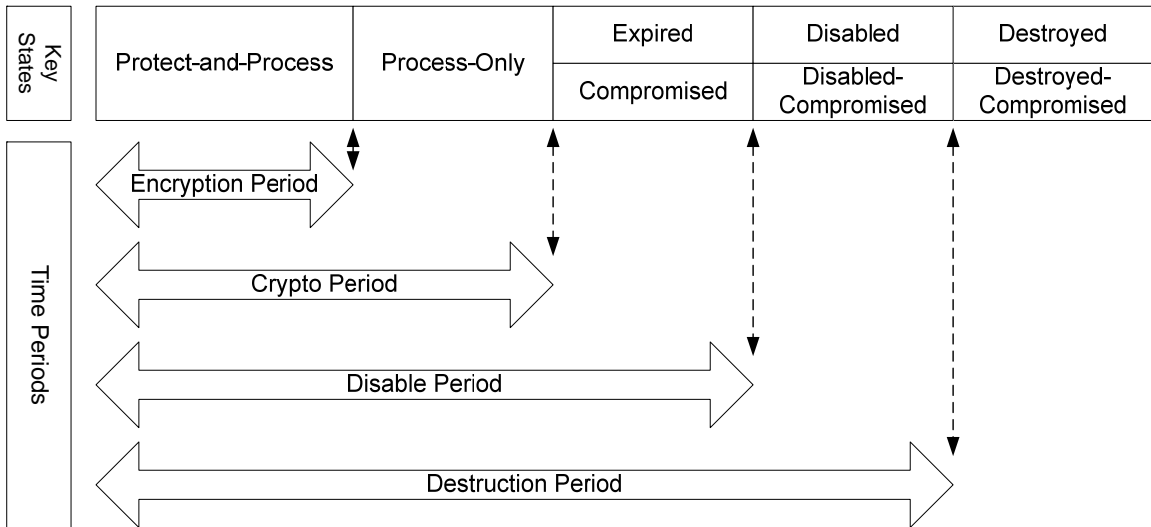
6 The key life cycle is based on four time periods:

- 7 — Encryption Period: This is the period of time after a key is activated that it can be used to encrypt
 8 data.
- 9 — Crypto Period: This is the period of time after a key is activated that it can be used for routine
 10 decryption. This time period should always be as long as or longer than the Encryption
 11 Period.
- 12 — Disable Period: This is the period of time after a key is activated before a key shall be disabled and
 13 become unavailable to KM Clients. A key may still be delivered to a KM Client after the
 14 Crypto Period ends but before the Disable Period ends, however, this may require
 15 administrative action on the KM Server. This case would be for a non-routine usage of the
 16 key. This time period should always be as long as or longer than the Crypto Period.
- 17 — Destruction Period: This is the period of time after a key is activated before the key is destroyed by
 18 the KM Server. This time period should always be as long as or longer than the Disable
 19 Period.

20 All key related time periods start when a key is activated; this occurs when a key is given to a KM Client
 21 by a KM Server, or when a key generated by a KM Client is given to the KM Server. Any time period can
 22 range from zero to forever, with the limitation that each time period must be at least as long as its
 23 predecessor as specified above.

24 Figure 4 illustrates the key related time periods and their relationship to the keys states of the key lifecycle
 25 model.

26 *[Mention that these are not the only time periods available – these are the time periods that can cause a*
 27 *key state transition.]*



28

29

Figure 4 Key Time Periods Compared to Key States

1 **4.5 Key Management Sequence Models**

2 *[Content TBD]*

3 **4.6 Key Management Operations Model**

4 Another important aspect of this key management standard is the conformant operations that are defined to
 5 take place between a key management client (KM Client) and a key management server (KM Server). A
 6 key management operation model is presented here to illustrate the high level flow of management
 7 information between these KM Client and KM Server entities. The general sequence of operations, notions
 8 of synchronous versus asynchronous operations, and support for single threaded versus multi-threaded
 9 operations are critical to the interoperability of conformant implementations of this standard.

10 *[More text and diagrams in development.]*

11 **4.7 Key Management Object Model**

12 To properly perform key management operations, a series of key management objects and their
 13 relationships must be defined to ensure interoperability of conformant implementations of this standard.
 14 Key management objects include all data structures and information that are necessary to provide complete
 15 key management services for encryption of stored data applications.

16 *[Need D1 Word source to copy object model definitions here?]*

17 **5. Key Management Namespace**

18 **5.1 Globally unique identifier for Key Management Services**

19 **5.1.1 Overview**

20 Each security object stored within the KMS shall be associated with a security object global unique
 21 identifier (SO_GUID), as defined in this sub clause.

22 A SO_GUID shall contain the following information, in order:

- 23 — **SO_Family**: A mandatory two-alphanumeric character code that describes the format for the
 24 following fields
- 25 — **SO_Domain**: An optional Fully qualified domain name as defined in RFC 1034
- 26 — **SO_Context**: An optional value that identifies a name space that is common across a set of keys
- 27 — **SO_Handle**: A mandatory value that is unique under the given SO_Context and corresponds to a
 28 specific KMS security object

29 The SO_Family shall be a value from Table A:

30 **Table 1 SO_Family values**

SO_Family	Description	Name Authority	Ref.
'km'	A format based on a URI name space	ICANN	5.1.2
'na'	A format based on an IEEE OUI name space	IEEE OUI	5.1.3
'rn'	A format based on 32 octet random numbers	None	5.1.4
'ek'	A format based on the EKMI name space	None	5.1.5
'00'	A SO_GUID that does not match any object	None	5.1.6

31

1 Each format from Table A shall support a unique serialization consisting of the concatenation of the four
 2 components required to create a SO_GUID. The total length of the SO_GUID shall not exceed 1024
 3 octets. The length of the SO_Handle shall be at least one octet.

4 The SO_GUID shall be constructed in a manner such that it is possible to unambiguously extract each of
 5 the four fields. See Annex E for more information.

6 A valid SO_GUID is a unique identifier that resolves to a KMS security object, a KMS symlink, or
 7 nothing. Security objects are the fundamental objects maintained by KMS. The KMS security objects
 8 include keys, KM Client information, key groups, key groups, KM policies, and KMS logs. The value of
 9 KMS symlink is a SO_GUID. Since it is possible that a SO_GUID refers to an object that does not exist
 10 (or that will exist in the future), a SO_GUID may resolve to nothing.

11 When a KM Server first resolves to a KMS symbolic link, it shall treat the value of the KMS symlink as a
 12 SO_GUID. The KM Server shall next attempt to resolve the new SO_GUID. To prevent infinite loops, the
 13 server may limit the number of times it resolve a KMS symbolic links for a given initial SO_GUID. The
 14 limit is KM Server configuration value that must be a minimum of 7. If a KM Server resolves an initial
 15 SO_GUID through too many KMS symlinks, it shall return a KMS symlink error to the KM Client
 16 indicating that the initial SO_GUID is not resolvable. If a KM Server detects that a SO_GUID can never
 17 resolve, say because the KMS symlink value is the same value as the SO_GUID of the symbolic link or
 18 because the KMS symlink is an invalid SO_GUID, then KM Server may return a KMS symlink error early.

19 The transport layer shall be able to determine the over-all length of the SO_GUID.

20 **5.1.2 km SO_Family: KMS Globally Unique ID using URI format with ICANN naming** 21 **authority**

22 The km SO_Family consists of URIs that are based on RFC 3986. When fully qualified they provide
 23 globally unique identifiers for security objects under key management.

24 URI syntax definition uses augmented Backus-Naur form (ABNF) as defined by RFC 4234 for
 25 descriptions.

26 If the SO_Family field of the SO_GUID is set to “km”, then the following requirements apply.

27 The SO_GUID shall consist of the following components:

28 **Table 2 SO_GUID Format for SO_Family ‘km’**

```

;;
;; SO_GUID (km SO_Family GUID in URI form)
;;
SO_GUID = SO_Family "://" SO_Domain SO_Context SO_Handle / SO_Directory

;;
;; SO_GUID components
;;

SO_Family = "km"

SO_Domain = KM_FQDN / dash

SO_Context = slash Object_Space Path

SO_Handle = POSIX_handle / non_POSIX_encoded_handle

SO_Directory = SO_Family "://" SO_Domain SO_Context

```

```

;;
;; SO_Domains (globally unique SO_Context)
;;

KM_FQDN = FQDN_non_dot 1*253FQDN_octet FQDN_non_dot

FQDN_non_dot = ALPHA / DIGIT / FQDN_encoded_non_dot

FQDN_octet = RFC1034_octet / FQDN_encoded_octet

RFC1034_octet = ALPHA / DIGIT / dot

FQDN_encoded_octet = FQDN_encoded_non_dot / dot

; FQDN_encoded_non_dot _ encoding of [^A-Za-z0-9.] (not dot nor RFC 1034 chars)
FQDN_encoded_non_dot = ( "%" ( "0" / "1" / "2" ) HEX
) / ( "2" ( DIGIT / "A" / "B" / "C" / "D" / "F" )
) / ( "3" ( "A" / "B" / "C" / "D" / "E" / "F" )
) / "%40"
/ ( "%5" ( "B" / "C" / "D" / "E" / "F" )
) / "%60"
/ ( "%7" ( "B" / "C" / "D" / "E" / "F" )
) / ( "%" ( "8" / "9" / "A" / "B" / "C" / "D" / "E" / "F" ) HEX )

;;
;; Object Spaces (part of the SO_Context)
;;

Object_Space = Object_Type / FQDN_Space / Family_Space / Reserved_Space

Object_Type = (ALPHA / DIGIT) *254POSIX_octet

FQDN_Space = dot KM_FQDN

Family_Space = dash 2(ALPHA / DIGIT)

Reserved_Space = 2dot *254POSIX_octet

;;
;; Paths (part of the SO_Context)
;;

Path = *(slash Directory) slash

Directory = (ALPHA / DIGIT) *254POSIX_octet

;;
;; SO_Handles
;;

POSIX_handle = (ALPHA / DIGIT) *254POSIX_octet

non_POSIX_encoded_handle = non_POSIX_encoded_first_octet
*254non_POSIX_next_octet

non_POSIX_encoded_first_octet = ALPHA / DIGIT / POSIX_encoded_first

POSIX_octet = POSIX_non_dot_octet / dot

POSIX_non_dot_octet = ALPHA / DIGIT / underscore / dash

;;

```

```

;; POSIX and non_POSIX encoding of Handles
;;

; POSIX_encoded_first _ encoding of [^A-Za-z0-9] (not Alphanumeric chars)
POSIX_encoded_first = ( "%" ( "0" / "1" / "2" ) HEX
) / ( "3" ( "A" / "B" / "C" / "D" / "E" / "F" )
) / "%40"
/ ( "%5" ( "B" / "C" / "D" / "E" / "F" )
) / "%60"
/ ( "%7" ( "B" / "C" / "D" / "E" / "F" )
) / ( "%" ( "8" / "9" / "A" / "B" / "C" / "D" / "E" / "F" ) HEX )

non_POSIX_next_octet = ALPHA / DIGIT / dash / dot / underscore /
non_POSIX_encoded_octet

; non_POSIX_encoded_octet _ encoding of [^A-Za-z0-9._-] (not POSIX chars)
non_POSIX_encoded_octet = ( "%" ( "0" / "1" ) HEX
) / ( "2" ( "A" / "B" / "C" / "F" )
) / ( "3" ( "A" / "B" / "C" / "D" / "E" / "F" )
) / "%40"
/ ( "%5" ( "B" / "C" / "D" / "E" )
) / "%60"
/ ( "%7" ( "B" / "C" / "D" / "E" / "F" )
) / ( "%" ( "8" / "9" / "A" / "B" / "C" / "D" / "E" / "F" ) HEX )

;;
;; Special characters and character sets (as single octets)
;;

ALPHA = %x41-5A / %x61-7A ; [A-Za-z]

DIGIT = "0" / "1" / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"

HEX = DIGIT / "A" / "B" / "C" / "D" / "E" / "F"

dash = "-" ; 0x2D

dot = "." ; 0x2E

slash = "/" ; 0x2F

underscore = "_" ; 0x5F

```

1

2 The maximum length of the SO_GUID shall be 1024 octets even though the above ABNF does not impose
3 such a limit on SO_GUID. The API shall return an error if SO_GUID length exceeds 1024 octets.

4 The SO_Domain shall not begin nor end with a dot. The maximum length of the SO_Domain shall be 255
5 octets (the limit currently imposed by RFC1034) even though the above ABNF does not impose such a
6 limit on KM_FQDN. The API shall return an error if SO_Domain length exceeds 255 octets.

7 The SO_Domain with the exception of the tri-graph encoded octets is defined by RFC1034. The dash
8 SO_Domain refers to a private domain. All other SO_Domain values are globally unique.

9 The maximum length of the SO_Context shall be 512 octets even though even though the above ABNF
10 does not impose such a limit on the SO_Context. The API shall return an error if the SO_Context length
11 exceeds 512 octets.

12 The Family_Space is used to map a SO_GUID from a non-“km” SO_Family to the “km” SO_Family. The
13 two octets following the dash shall be a valid SO_Family other than “km” or “00”. The Family_Space “-

1 km” is not a valid Object_Space in the “km” SO_Family. The Family_Space “-00” is not a valid
 2 Object_Space in the “km” SO_Family. When a valid SO_GUID in the km SO_Family has an
 3 Object_Space that is a Family_Space, the SO_Handle shall be the SO_GUID of the referenced SO_Family,
 4 is a KMS symlink whose value is the SO_Handle.

5 An SO_GUID in the km SO_Family with an “r” Object_Space shall always refer to a KMS record ID
 6 whose value is 12 octets long. The record ID provides a unique fixed length identifier within a
 7 SO_Domain. It may be used in place of the SO_GUID for lookup and retrieval of keys. Record ID’s are
 8 assigned by a KM Server so as to be unique throughout the SO_Domain.

9 For every security object that exists in a SO_Domain in the km SO_Family, there shall exist a SO_GUID in
 10 the “r” Object_Space.

11 The Reserved_Space is reserved for future use by this standard.

12 The SO_Context is unique within its SO_Domain.

13 The maximum length of the SO_Handle shall be 255 octets even though even though the above ABNF does
 14 not impose such a limit on the SO_Handle. The API shall return an error if the SO_Handle length exceeds
 15 255 octets.

16 The SO_Handle is a unique value within a given SO_Context that is under a given SO_Domain. The
 17 SO_Handle is a KM filename. A SO_Handle identifies a security object (e.g., a key, policy, key group, key
 18 group, client, log, record number, etc.)

19 NOTE—No character from a valid POSIX portable filename shall be encoded as a hexadecimal tri-graph.

20 When used in fully qualified SO_GUID for a non-directory security object, the SO_Handle follows the
 21 SO_Context. The SO_GUID of a directory security object has no SO_Handle. The SO_GUID of a
 22 directory security object shall end in a slash.

23 The Object_Space values for the Security Object Types defined in this standard shall be according to the
 24 following table:

25 **Table 3 URI Required Object_Space values for various Security Objects**

Security Object Type	Object_Space value
Key	“key”
KMS Policy	“policy”
Key Management Realm	“realm”
Key Group	“group”
KM Client	“client”
KMS Log	“log”
Record ID	“r”
PKCS#11	“p11”

26

27 Security objects with may be referenced in several ways:

- 28 1) Globally by: SO_Family, SO_Domain, SO_Context and SO_Handle
- 29 2) SO_Context and SO_Handle within the clients current SO_Domain
- 30 3) SO_Handle within a clients current SO_Context

31 The following are examples of a fully qualified SO_GUID in the km SO_Family:

32 Key

- 1 1) km://example.org/key/dir1/dir2/key123
 2 2) km://example.com/key/dir1/%00%00%EA%05
 3 3) km://traders.bigbank.com/key/000102030405060708090A0B0C0D0E0F
 4 Policy
- 5 4) km://example.net/policy/storsecpolicy/kmspolicy/keypolicy3
 6 Client
- 7 5) km://subdomain.example.com/client/location/kmclient1
 8 Realm
- 9 6) km://division.bigcompany.example.com/realm/San%20Jose
 10 Record ID
- 11 7) km://example.biz/r/F0F1F2F3F4F5F6F7F8F9FAFB
 12 8) km://example.biz/r/%00%01%02%03%04%05%06%07%08%09%0A%0B
 13 FQDN_Space
- 14 9) km://dc4.bigfirm.example.biz/.prod.acme.com/acme_legacy_key1
 15 10) km://dc4.bigfirm.example.biz/.old.example.com/key/symlink1
 16 Family_Space
- 17 11) km://prod.example.info/-rn/legacy/server1/%02%32%09

18 **5.1.3 na: SO_Context and SO_Handle using IEEE OUI name address authority**

19 If the SO_Family field of the SO_GUID is set to 'na', then the following requirements apply:

- 20 a) The SO_Context field of the SO_GUID shall contain an 8-octet name address authority (NAA)
 21 identifier that identifies the KM Server that created the security object
- 22 b) The SO_Handle field of the SO_GUID shall contain a variable-length binary number that uniquely
 23 identifies this object within the scope of the NAA within the SO_Context field.

24 Table D shows the format of a SO_GUID when the family is set to 'na'.

1

Table 4 Format of a SO_GUID with a family set to 'na'

Bit										
Octet	7	6	5	4	3	2	1	0		
0	'n' (6E16)									
1	'a' (6116)									
2	NAA									
...	NAA specific									
9										
10	(MSB)									
n-1	SO_Handle							(LSB)		

2

3 If the SO_GUID length is less than 11 octets, then the GUID parser shall return FAIL. Otherwise, the
 4 parser shall return SO_Context and the NAA specific field forms the NAA identifier as defined by
 5 T10/SPC-4.

6 The NAA field shall contain one of the values given in Table 5.

7

Table 5 Values for the NAA field

Value	Description	Name Authority	Reference
2	IEEE Extended	IEEE OUI	T10/SPC-4
3	Locally assigned	None	T10/SPC-4
5	IEEE Registered	IEEE OUI	T10/SPC-4
All others	Reserved		

8

9 If the NAA field is set to 3 (e.g. locally assigned), then an NAA specific value of all zeros indicates that
 10 there is no valid context for this SO_GUID and that the SO_Handle is the only information that identifies
 11 the security object.

12 If the NAA field value is not contained within Table E, then the parser shall return FAIL.

13 When using an NAA field value of 3 (e.g. locally assigned), then there is no assertion that this SO_GUID is
 14 globally unique. The KM Server should take care when migrating such security objects to ensure that the
 15 set of SO_GUIDs in the originating system do not collide with SO_GUIDs from the destination system.

16 The NAA specific field shall contain data as defined by the reference associated with the corresponding
 17 NAA field value.

18 The SO_Handle field shall contain an application-specific identifier to a particular security object that is
 19 unique within the scope of the NAA identifier. The SO_Handle shall contain at least 1 octet and no more
 20 than 64 octets. Otherwise, the SO_GUID parser shall return FAIL.

1 **5.1.4 rn: SO_Handle using 1 to 64 octet random numbers and no naming authority**

2 If the SO_Family field of the SO_GUID is set to 'rn', then the following requirements apply:

3 a) The SO_Handle field of the SO_GUID shall be a 1 to 64 octet binary.

4 The “rn” SO_Family is a SO_Handle only address space. It has only one SO_Domain and SO_Context.

5 **5.1.5 ek: SO_Context and SO_Handle using Enterprise Key Management Infrastructure** 6 **(EKMI)**

7 If the SO_Family field of the SO_GUID is set to “ek”, then the following requirements apply. The
8 SO_GUID shall consist of the following components:

9 <SO_GUID> = <SO_Family> “:/" <SO_Handle> /

10 a) <SO_Family> = “ek”

11 • “ek” shall be followed by “:/" when preceding a <SO_Handle>

12 b) <SO_Handle> = <GKID>

13 • <GKID> = <DID> 0x2D <SID> 0x2D <KID>

14 • <DID> = DIGIT 1*8

15 • <SID> = DIGIT 1*8

16 • <KID> = DIGIT 1*8

17 • The maximum length of the <SO_Handle> shall be 27 octets. The API shall return an error if the
18 length exceeds 27 octets.

19 • A SO_Handle identifies a security object - e.g., key

20 Valid examples of the SO_Handle within the “ek” namespace are as follows:

21 • 0-0-0

22 • 1-2-3

23 • 10514-22-344342232

24 • 18446744073709551615-18446744073709551615-18446744073709551615

25 *[Content TBD]*

26 **5.1.6 00: A SO_GUID that does not match any security object**

27 This SO_Family has only one valid SO_GUID: the two octets “00”.

28 This SO_Family allows an interface return a SO_GUID that is valid but does not match any security object.
29 One may use the “00” SO_GUID as a sentinel to terminate a list of SO_GUIDs.

30 **6. Key Management Objects**

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

4 **6.1 Key Object**

5 **Scope:** Client & Server.

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

7 **6.1.1 Attributes**

8 A key object distributed by a KM Server contains the following attributes:

9 — **KEY_ID** (Type: SO_GUID)

10 — **FRIENDLY_NAME** (Optional: Type:String) A non-unique identifier within the KMS. Additional
 11 attributes may be required to make a unique reference.

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

14 — **STATE** (Type:String) See Section 4.4.1 for definition section

15 — **T_EXPIRED** (Type: UTC - time beyond which the key should not be used to encrypt new data)

16 — **T_DISABLED** (Type: UTC - time beyond which the key should not be used)

17 — **T_CACHED** (Type: 64-bits – seconds that the key may be cached for. This may be differentiated
 18 by endpoint)

19 — **CIPHER_TYPE** (Type:String OID) (see Annex D)

20 — **KEY_BLOB** (Object as defined in section 6.2) (see 3.2.22)

21 — Vendor_specific_extensions

22 — Application_extensions

23 — (Versioning_information:)

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

32 — version (Type: unsigned Numeric)

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

34 — **EDIT_DATETIME** (Type: DATETIME)

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

36 In addition, a KM Server must be capable of representing the following attributes and references:

37 — Realm_Associations

38 NOTE—keys shall not be delivered to endpoints without compatible Realm rights)

39 — Wrapping_Policy

- 1 — description (Type:String)
- 2 — Source
- 3 — Represents the identity of the originator of the key value. (a specific KM Server, a specific KM
- 4 Client, a specific CU)
- 5 — ATTRIBUTE_ASSOCIATIONS
- 6 — (A list of named value pairs useable as an alternate mechanism to define or reference a key . Keys
- 7 can be retrieved by their key_ID or alternatively by an ANDED match on these NVPs)
- 8

9 In addition, a KM Server must be capable of representing the following associations:

- 10 — use_by_clients

11 NOTE—does not alter Versioning_Information for a key itself. Note since any given key may be used by
 12 multiple clients or CUs, this tracking must be maintained, so the KM Server is capable of initiating
 13 unsolicited updates to a KM Client when a key's attributes or policies change)

- 14 — USE_BY_CUS (see above note)

15 6.1.2 States

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

17 6.1.3 Operations

- 18 — Create
- 19 — Get
- 20 — Store

21 6.2 Key Blob

22 6.2.1 Attributes

- 23 — ProtocolVersion (Type:int and defined as 1 for this version of the standard – This attribute shall
- 24 remain constant for all clients for this version of the standard.).
- 25 — WRAPPING_TYPE (Type:String) – and can take any of the values as listed in the key wrapping
- 26 section.
- 27 — Length (Type:int)
- 28 — Data (Type: Character Array)

29 6.3 Key_Template

30 **Scope:** Server.

31 Narrative Text: This object is not transmitted over the wire between the client and the server. The only
 32 attribute that a client needs to understand is the notion of 'key_template_id' which can be passed in as a
 33 Dataset attribute.

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

1 It should be possible to make a key creation request “byTemplate”, with or without additional dataset
 2 bindings. It is not possible to make a key retrieval request “byTemplate” without distinguishing dataset
 3 bindings.

4 **6.3.1 Attributes**

5 A Key_Template object defined within a KM Server contains the following attributes:

6 — **KEY_TEMPLATE_ID** (Type: SO_GUID)

7 — **FRIENDLY_NAME** (Optional: Type:String) Unique within a KM Server

8 NOTE—It should be possible to request a key creation by template using its Friendly_name

9 — **CIPHER_TYPE** (Type:String OID – **TODO:** Insert)

10 — Vendor_specific_extensions

11 — Application_extensions

12 — Caching_Policy

13 — (Versioning_information:)

14 — *VERSION* (Type: unsigned Numeric)

15 — *edit_Datetime* (Type: DATETIME)

16 — Realm_Associations

17 — Wrapping_Policy

18 — *DESCRIPTION* (Type:String)

19 **6.4 ENDPOINT_TYPE**

20 Endpoint_Type is an object to simplify the need to exchange capabilities between a KM Client or CU and a
 21 KM Server, as well as managing a collection of capabilities at the Server. (Discussion point: It would be
 22 desirable if these values were standardized in some registry.) An Endpoint_Type shall always equate to a
 23 deterministic set of capabilities, though the converse need not be true. During registration, KM Clients or
 24 CUs shall present identifying information that will allow a KM Server to map it to an Endpoint_Type.

25 **6.4.1 Attributes**

26 — **ENDPOINT_TYPE_ID** (Type:TBD)

27 — **CAPABILITIES** (Note: common registry should allow retrieval of such characteristics as has-
 28 certificate, understands-time, min and max keyID lengths, never-exposes-key, hashSM, etc.)

29 **6.5 REALM (optional)**

30 Realms are used to segment objects into separate administrative domains.

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

37 When a KM Server provides Realm support, the KM Server must insure no object is assessable unless the
 38 requesting endpoint or administrator has been properly associated with an incorporating realm that allows

1 the access. Realms must allow the following distinctions: create, edit, delete, read, reference (use but not
2 view). The most privileged right from any associated realm may be use to determine an access.

3 **6.5.1 Attributes**

- 4 — REALM_ID (Type: *TBD*; where domain is a string that is in compliance with DNS name as
5 defined by RFC 1034.
- 6 — DESCRIPTION

7 **6.6 CU (Cryptographic Unit)**

8 Scope: Client & Server

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

12 **6.6.1 Attributes**

- 13 — CU_ID (Type:SO_GUID)
- 14 — Array of Name/Value pairs.

15 NOTE—The name of the attributes that are part of the CU object shall follow the following convention:

16 Example:

17 `km://example.com/client/client_path/km_client1_name/CU_ID/attribute_name`

18 In addition the SO_Domain/SO_Context combination `ieee.org/client/siswg/` is reserved for use by this
19 standard.

- 20 — Realm_Associations (Server Only)
- 21 — Endpoint_type_ID (type:TBD Server Only.)

22 NOTE—KM Clients shall provide a CIM_PhysicalElement or CIM_SoftwareIdentity object upon
23 registration of a CU, which will allow a KM Server to map an entity to a TYPE_ID. Or perhaps we define
24 a new CIM object derived from common ones to include capabilities we want that may not be determined
25 by attributes in the mentioned CIM objects.

- 26 — Client_associations (Server Only)

27 NOTE—This might be done with a Client_Group. A CU shall initially be associated with full rights granted
28 to its registering KM Client. Other behaviors to manage associations is outside the scope of this spec., but
29 a KM Server must be capable of conforming to a policy or configuration that prevents a CU from receiving
30 its key from an “unauthorized” client. This can easily be accomplished by restricting access to keys both to
31 authorized KM Clients and authorized CUs.)

- 32 — AUTHENTICATION_POLICY
- 33 — AUTHENTICATION_VALUES (It must be possible for a CU to authenticate to the KM Server
34 through an untrusted KM Client)
- 35 — List of {CREDENTIAL_LENGTH,CREDENTIAL_VALUE} tuples.
- 36 — WRAPPING_POLICY

37 NOTE—this may typically be inherited via endpoint_type_id

- 1 — *WRAPPING_VALUES* (Since CUs can uniquely protect some key wrapping values, such as private
2 keys, data can pass through a KM Client without exposure)

3 **6.7 KM Client**

4 **Scope:** KM Client & KM Server.

5 The KM Client object consists of its credentials and capabilities.

6 **6.7.1 Attributes**

- 7 — *CREDENTIAL_TYPE* (Session, Username/Password, Symmetric / Asymmetric key, SSO, CHAP)
8 — List of {*CREDENTIAL_LENGTH*,*CREDENTIAL_VALUE*} tuples.
9 — *Realm_Associations*
10 — *Endpoint_Type_ID*
11 — *WRAPPING_POLICY*

12 **6.7.2 States**

- 13 — Active
14 — Disabled/Locked
15 — Authenticated

16 **6.7.3 Operations**

- 17 — Create
18 — Delete
19 — Authenticate
20 — Disable

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

23 **6.8 Capability**

24 **6.8.1 Attributes**

- 25 — Name (Type: String)
26 — Supports AES 128
27 — Supports AES 256
28 — Supports 3DES
29 — Understands Relative time (elapsed time)
30 — Understands Universal time
31 — Includes HSM
32 — Includes TPM-
33 — min and max keyID lengths
34 — never-exposes-key
35 — is client-cu Hardware combo

- 1 — is_Kernel Software
- 2 — is User Space Software
- 3 — is Hardware
- 4 — Utilizes Host for Crypto
- 5 — can Persistently Cache keys
- 6 — Offers api to provide key in clear
- 7 — .. and more tbd

8 **6.8.2 States**

9 None.

10 **6.8.3 Operations**

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

13 **6.9 Data Sets**

14 **Scope:** Client, CU & Server.

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

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

20 **6.9.1 Attributes**

- 21 — NAME
- 22 — VALUE
- 23 — SIZEOF_VALUE

24 **6.10 Client Groups**

25 **Scope:** Server only.

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

28 **6.10.1 Attributes**

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

33 **6.11 Key Groups**

1 **Scope:** Server only.

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

4 **6.11.1 Attributes**

5 — TYPE (STATIC or DYNAMIC)

6 — List of CLIENT_SO_GUIDs (only in case of static binding)

7 — List of {PATTERN, DATASET} tuple.

8 — REALM_ASSOCIATIONS

9 **7. Key Management Policies**

10 This clause only applies to the KM server and does not represent objects sent between the KM server and
11 the KM client. These objects may be represented in KM Server to KM Server operations.

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

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

16 **7.1 Key Assignment Policy**

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

20 A key assignment policy determines

21 — the type of unencrypted data that is determined to be encrypted with a specific key.

22 — how often to generate new keys

23

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

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

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

32 container attribute name = "storage_server_name" value="Ireland.com"

33 data attribute name = "financial"

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

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

8 **7.1.1 Attributes**

- 9 — KEY_ASSIGNMENT_POLICY_ID
- 10 — DESCRIPTION
- 11 — REALM_ASSOCIATIONS

12 **7.2 Retention Policy**

13 **7.2.1 Overview**

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

17 This policy should be superseded by the key life cycle.

19 **7.2.2 Attributes**

- 20 — RETENTION_POLICY_ID
- 21 — DESCRIPTION
- 22 — REALM_ASSOCIATIONS
- 23 — T_EXPIRATION
- 24 — T_DISABLE

25 **7.2.3 States**

- 26 — Created
- 27 — Assigned
- 28 — Enforcing
- 29 — Disabled

30 **7.2.4 Operations**

- 31 — Add
- 32 — Associate
- 33 — Disassociate
- 34 — Delete

35 **7.3 Wrapping Policy**

1 The wrapping policy indicates whether a key should be wrapped prior to being dispatched to a client. This
 2 policy may be referenced by key, group, CU, Endpoint_Type, or KM Client. If multiple policies are
 3 defined then the order of precedence shall be per the following:

- 4 1. Key shall prevail over KeyGroup
- 5 2. Key or KeyGroup shall prevail over CU
- 6 3. CU shall prevail over KM Client's Endpoint_Type
- 7 4. KM Client shall prevail over KM Client's Endpoint_Type
- 8 5. If both a CU and Client specify (or inherit) a Wrapping Policy, the key shall be double wrapped,
 9 first by policy prevailing for the CU, then by prevailing ClientPolicy. The KM Client shall
 10 unwrap the key and pass the wrapped Client_Key for subsequent unwrapping.

11 7.3.1 Attributes

- 12 — WRAPPING_TYPE (integer)
- 13 — WRAPPING_MODE (integer)

14 7.3.2 States

- 15 — Created
- 16 — Assigned
- 17 — Enforcing
- 18 — Disabled

19 7.3.3 Operations

- 20 — Add
- 21 — Associate
- 22 — Disassociate
- 23 — Delete

24 7.4 Audit Policy

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

26 *[Content TBD]*

27 7.4.1 Attributes

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

31 NOTE—The only mandatory type that should be supported is 'log' and the mandatory dispatch destinations
 32 are local and syslog.

33 7.4.2 States

- 1 — Created
- 2 — Assigned
- 3 — Enforcing
- 4 — Disabled

5 **7.4.3 Operations**

- 6 — Add
- 7 — Associate
- 8 — Disassociate
- 9 — Delete

10 **7.5 Access/Distribution Policy**

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

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

16 This policy is referenced by a key or key group.

17 NOTE—this policy governs what endpoints MAY receive a key, but cannot be used to determine which
 18 endpoints in fact received any given key.

19 **7.5.1 Attributes**

- 20 — SO_GUID
- 21 — Client or clientGroup list
- 22 — CU list
- 23 — KM server list.
- 24 — REALM_ASSOCIATIONS

25 **7.5.2 States**

- 26 — Created
- 27 — Assigned
- 28 — Enforcing
- 29 — Disabled

30 **7.5.3 Operations**

- 31 — Add
- 32 — Associate
- 33 — Disassociate
- 34 — Delete

1 **7.6 Caching Policy**

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

4 **7.6.1 Attributes**

5 — CACHING_TYPE, T_CACHE_INTERVAL tuples (list)

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

8 — HARDWARE TYPE – (e.g., HSM, TPM, FIPS 140-2 Approved, software)

9 — DESTINATION (particular clients, where it is stored in a file, is it unwrapped in memory)

10 — STANDARDS CONFORMANCE (FIPS 140-2, VISA PCI,...)

11 — CACHE_TIME – Number of seconds the CU is allowed to cache a particular key

12 — USAGE_COUNT – How many times can the CU use the key before purging from cache (do we
13 need this – general feeling was that it's not generally applicable, but could be used with credit
14 cards)

15

16 Applicable objects

17 • Key Object

18 • Key Group Object

19 **7.6.2 States**

20 — Created

21 — Assigned

22 — Enforcing

23 — Disabled

24 **7.6.3 Operations**

25 — Add

26 — Associate

27 — Disassociate

28 — Delete

29 **8. Key Management Operations**

30 **8.1 Register KM Client**

31 Scope: KMCS Ops, including registration for KM Client or CU

1 **8.1.1 Overview**

2 This operation allows clients to register themselves with a KM server. These clients by default, should be
3 placed in an administrative realm where they are not allowed to perform any operations unless an
4 administrator approves their transition from an ‘unapproved’ state to an ‘approved’ state by verifying the
5 authentication credentials that have been provided to by the client.

6 This behavior may be modified based on the trust that a KM server has on the credentials that are being
7 presented to it. Examples of such trust might include CA signed certificates, trusted Kerberos realm, etc.

8 Though support for this operation is mandatory, a KM server might require an administrative action to
9 enable this operation. Based on the threat model in which the server is deployed, a KM server shall provide
10 an administrator the ability to disable this operation.

11 **8.1.2 Input / Output / Error**

- 12 — (I): Client
- 13 — (I): Credentials
- 14 — (O): Boolean (SUCCESS OR FAILURE)
- 15 — (E): E_SELFREGISTRATION_UNSUPPORTED.

16 **8.2 Authenticate**

17 Scope: KMCS

18 **8.2.1 Overview**

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

22 **8.2.2 Input / Output / Error**

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

27 **8.3 Capability Negotiation**

28 Scope: KMCS

29 **8.3.1 Overview**

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

32 **8.3.2 Input / Output / Error**

- 33 — (I): Client
- 34 — (I): List of Capability Objects

1 — (O): List of Capability Objects that are supported by the KM server

2 **8.4 Get Server Capabilities**

3 — (I): Client

4 — (O): List of Capability Objects.

5

1 **8.5 Create/Generate Key**

2 Scope: KMCS, KM Server

3 **8.5.1 Overview**

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

7 **8.5.2 Input / Output / Error**

8 — (I): Client

9 — (I): *CU_ID* or EndPoint's CIM Object - Identifier of final destination for the key.

10 — (I): List of Dataset objects.

11 — (O): Key (including unique So_Guid)

12 — (E): ...

13 **8.6 Store Key**

14 Scope: KMCS, KM Server

15 **8.6.1 Overview**

16 Keys that are generated at the KM Client can be stored in the KM Server by invoking its store
17 functionality.

18 **8.6.2 Input / Output / Error**

19 — (I): Client

20 — (I): List of Dataset objects.

21 — (I): *CU_ID* or EndPoint's CIM Object - Identifier of final destination for the key.

22 — (O): *Key_SO_GUID*

23 — (I): *Friendly_Name*

24 — (E): ...

25 **8.7 Get Key**

26 **8.7.1 Overview**

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

31 **8.7.2 Input / Output / Error**

32 — (I): Client

33 — (I): *CU*

- 1 — (I): List of Dataset objects.
- 2 — (O): Key
- 3 — (E): ...

4 **8.8 Push Audit Message**

5 **8.8.1 Overview**

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

9 **8.8.2 Input / Output / Error**

- 10 — (I): Client or CU
- 11 — (I): Key SO_GUID
- 12 — (I): Message (Optional)
- 13 — (O): Boolean – SUCCESS/FAILURE.
- 14 — (E): ...

15 **8.9 Get Random Bytes**

16 **8.9.1 Overview**

17 This operation is intended to be used by KM Clients that may want random bytes for undisclosed use or
18 need a seed from a trusted source for generating their own keys.

19 **8.9.2 Input / Output / Error**

- 20 — (I): Client
- 21 — (I): numbers of bytes desired
- 22 — (O): Base64 Encoded bytes

23 **8.10 GetStatus -- KM Client Service (optional) -- [Server initiated]**

24 **8.10.1 Overview**

25 A KM Server initiated message that requests the current status from a KM Client. This feature is

26 **8.10.2 Input / Output / Error**

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

1 **8.11 UpdatePending --KM Client Service (optional) [Server initiated]**

2 **8.11.1 Overview**

3 KM Server notification to KM Client of required updates.

4 KM Clients expose this service. KM Servers shall repeat this service until the client acknowledges
5 currency by virtue of matching UpdateVersioningTokens. Note, these tokens are used for sequencing
6 between this service and GetChangeList. A simple implementation of this would be for the KM Server to
7 maintain a VersioningToken to represent the latest version of “Everything” for a KM Client or CU and a
8 response to a GetUpdateList that returns everything.

9 **8.11.2 Input / Output / Error**

- 10 — (I): Type (Keys, AllKeys and custom types)
- 11 — (I): Scope: (KM Client or CU identifier)
- 12 — (I): UpdateVersioningTokens - KM Server sends its tokens representing the state to which the
13 client (or CU) needs to update.
- 14 — (O): UpdateVersioningTokens - KM Client sends its tokens representing the state it has received

15 **8.12 GetUpdateList -- KM Server Service [Client initiated]**

16 **8.12.1 Overview**

17

18 **8.12.2 Input / Output / Error**

- 19 — (I): Type: (Keys, AllKeys and custom types)
- 20 — (I): Scope: (KM Client or CU identifier)
- 21 — (I): UpdateVersioning Tokens (zero values shall result in all requested instances of requested
22 type)
- 23 — (O): Requested objects
- 24 — (O): New UpdateVersioningTokens

25 **9. Key Management Messaging**

26 **9.1 SOAP based protocol**

27 This section specifies the SOAP implementation for the KMS API described in section 5.

28 This is a Remote Procedure Call (RPC) interface exposed by a KM Server. Request and response messages
29 are formatted using the SOAP Document/Literal Wrapped pattern, and are transmitted over a TLS
30 encrypted connection using HTTP 1.1. This API complies with Web Services Interoperability
31 Organization's WS-I Basic Profile 1.0 (<http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html>).

32 The request and response message input and return values are taken from the key exchange structure
33 element.

34 An xml schema definition and WSDL are provided in Annex D.

1 An example key exchange structure is provided below:

```

<keyExchangeStructure>

  <keyObject>

    <keyID>
      <keyClassname> kcn.2007-05.org.IEEE.1619:XTS-AES-256 </keyClassName>
      <keyFormat> kfn.2007-05.org.IEEE.1619:generic</keyFormat>
      <keyIdSpace>example.com</keyIdSpace>
      <KeyIdValue>123456</keyIdValue>
    </keyID>

    <keyContents>
      <secretContents>SecretBinaryValue</secretContents>
    </keyContents>

    <keyOrigin>
      <generationTime>January 1, 2000</generationTime>
      <originatorID>kms1@example.com</originatorID>
      <requestorID>user1@example.com</requestorID>
    </keyOrigin>

  </keyObject>

  <dataSetBindings>

    <containerAttributes>
      <storagePoolLabel>Ireland</storagePoolLabel>
      <storageTapeLabel>0002</storageTapeLabel>
    </containerAttributes>

    <dataAttributes>
      <attribute name="customer data">
    </dataAttributes>

  </dataSetBindings>

  <keyAssignmentPolicy>
    <attribute name="storagePoolLabel" value="Ireland">
  </keyAssignmentPolicy>

  <replicationPolicy>
    <keyManagerID>kms2@example.com</keyManagerID>
    <commitment_request>notify</commitment_request>
    <Delay>20</Delay>
  </replicationPolicy>

  <retentionPolicy>
    <expirationTime>January 1, 2020</expirationTime>
    <disableTime>January 1, 2010</disableTime>
    <purgeTime>January 1,2021</purgeTime>
  </retentionPolicy>

</keyExchangeStructure>

```

2

3 9.1.1 getMethods request message

4 **Purpose:** Retrieves a list of valid RPC method names.

1 **Parameters:** None.

2 **Example:** getMethodsRequest Message

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:km="http://www.IEEE.org/1619/KeyManagement/">

  <soap:Body>
    <km:KeyManagement_GetMethods>
      </km:KeyManagement_GetMethods>
    </soap:Body>

</soap:Envelope>
```

3

4 **Return parameters:** The response contains a list of valid method names:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:km="http://www.decru.com/KeyManagement/">

  <soap:Body>

    <km:KeyManagement_GetMethodsResponse>
      <method>KeyManagement_GetKeyByStorageAttributes</method>
      <method>KeyManagement_GetKeyByID</method>
      <method>KeyManagement_UpdateKeyInfo</method>
      <method>KeyManagement_GenerateKey</method>
      <method>KeyManagement_GetSeed</method>
      ...
    </km:KeyManagement_GetMethodsResponse>

  </soap:Body>

</soap:Envelope>
```

5

6

7 **9.1.2 getKeyByStorageAttributes request message**

8 **Required Parameters:**

9 — DataSetBindings

10 All data set bindings that client understands must be provided – this list is set at client registration.

11 **Optional Parameters:**

12 — Key Classname and Key Format

13 The effect of supplying these attributes is to request that a specific key type be used, in the case that both
14 the encryption policy and the client understand multiple types

15 ReplicationPolicy

16 The effect of supplying these attributes is to ensure that the updated data set bindings have been replicated
17 prior to shipment of the key. In case a new key is generated with this command, it shall also be replicated.

1 **Example:**

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:km="http://www.IEEE.org/1619/KeyManagement/">

  <soap:Body>

    <km:KeyManagement_GetKeyByStorageAttributes>

      <kex_input>

        <dataSetBindings>

          <containerAttributes>
            <storagePoolLabel>Ireland</storagePoolLabel>
            <storageTapeLabel>0002</storageTapeLabel>
          </containerAttributes>

          <dataAttributes>
            <attribute name="customer data">
          </dataAttributes>

        </dataSetBindings>

      </kex_input>

    </km:KeyManagement_GetKeyByStorageAttributes>

  </soap:Body>

</soap:Envelope>

```

2

3 **Return parameters:** key exchange structure containing or SOAP Fault.

```

<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
xmlns:km="http://www.IEEE.org/1619/KeyManagement/">

  <soap:Body>

    <km:KeyManagement_GetKeyByStorageAttributesResponse>

      <kex_output>

        <keyExchangeStructure>

          <keyObject>

            <keyID>
              <keyClassname> kcn.2007-05.org.IEEE.1619:XTS-AES-256
            </keyClassname>
            <keyFormat> kfn.2007-05.org.IEEE 1619:generic
            </keyFormat>
            <keyIdSpace>example.com</keyIdSpace>
            <KeyIdValue>123456</KeyIdValue>
          </keyID>

          <keyContents>
            <secretContents>SecretBinaryValue</secretContents>
          </keyContents>

          <keyOrigin>

```

```

        <generationTime>January 1, 2000</generationTime>
        <originatorID>kms1@example.com</originatorID>
        <requestorID>user1@example.com</requestorID>
    </keyOrigin>

</keyObject>

<dataSetBindings>

    <containerAttributes>
        <storagePoolLabel>Ireland</storagePoolLabel>
        <storageTapeLabel>0002</storageTapeLabel>
    </containerAttributes>

    <dataAttributes>
        <attribute name="customer data">
    </dataAttributes>

</dataSetBindings>

<retentionPolicy>
    <expirationTime>January 1, 2020</expirationTime>
    <disableTime>January 1, 2010</disableTime>
    <purgeTime>January 1,2021</purgeTime>
</retentionPolicy>

<replicationPolicy>
    <keyManagerID>kms2@example.com</keyManagerID>
    <commitment_request>notify</commitment_request>
    <Delay>20</Delay>
</replicationPolicy>

<keyAssignmentPolicy>
    <attribute name="storagePoolLabel" value="Ireland">
</keyAssignmentPolicy>

</keyExchangeStructure>

</kex_output>

</km:KeyManagement_GetKeyByStorageAttributesResponse>

</soap:Body>

</soap:Envelope>

```

1 9.1.3 getKeyByID request message

2 **Purpose:** To obtain a key needed to decrypt encrypted data, or possibly to obtain a key needed to encrypt
 3 data, if the ID of the key is known.

4 **Required parameter:**

5 — Key ID portion of the Key element

6

1 **Example:**

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:km="http://www.IEEE.org/1619/KeyManagement/">

  <soap:Body>

    <km:KeyManagement_GetKeyByID>

      <kex_input>

        <keyID>
          <keyClassname> kcn.2007-05.org.IEEE.1619:XTS-AES-256
          </keyClassname>
          <keyFormat> kfn.2007-05.org.IEEE 1619: generic</keyFormat>
          <keyIdSpace>example.com</keyIdSpace>
          <keyIdValue>123456</keyIdValue>
        </keyID>

      </kex_input>

    </km:KeyManagement_GetKeyByID>

  </soap:Body>

</soap:Envelope>

```

2

3 **Return parameters:** key element or SOAP Fault.

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:km="http://www.IEEE.org/1619/KeyManagement/">

  <soap:Body>

    <km:KeyManagement_GetKeyByIDResponse>

      <kex_output>

        <keyExchangeStructure>

          <keyObject>

            <keyID>
              <keyClassname>kcn.2007-05.org.IEEE.1619:XTS-AES-256
              </keyClassname>
              <keyFormat>kfn.2007-05.org.IEEE 1619:generic</keyFormat>
              <keyIdSpace>example.com</keyIdSpace>
              <keyIdValue>123456</keyIdValue>
            </keyID>

            <keyContents>
              <secretContents>SecretBinaryValue</secretContents>
            </keyContents>

            <keyOrigin>
              <generationTime>January 1, 2000</generationTime>
              <originatorID>kms1@example.com</originatorID>
              <requestorID>user1@example.com</requestorID>
            </keyOrigin>

          </keyObject>

        </keyExchangeStructure>

      </kex_output>

    </km:KeyManagement_GetKeyByIDResponse>

  </soap:Body>

</soap:Envelope>

```

```

    <dataSetBindings>
      <containerAttributes>
        <storagePoolLabel>Ireland</storagePoolLabel>
        <storageTapeLabel>0002</storageTapeLabel>
      </containerAttributes>
      <dataAttributes>
        <attribute name="customer data">
      </dataAttributes>
    </dataSetBindings>
    <keyAssignmentPolicy>
      <attribute name="storagePoolLabel" value="Ireland">
    </keyAssignmentPolicy>
    <replicationPolicy>
      <keyManagerID>kms2@example.com</keyManagerID>
      <commitment_request>notify</commitment_request>
      <Delay>20</Delay>
    </replicationPolicy>
    <retentionPolicy>
      <expirationTime>January 1, 2020</expirationTime>
      <disableTime>January 1, 2010</disableTime>
      <purgeTime>January 1,2021</purgeTime>
    </retentionPolicy>
  </keyExchangeStructure>
</kex_output>
</km:KeyManagement_GetKeyByIDResponse>
</soap:Body>
</soap:Envelope>

```

1

2 **9.1.4 QueryKey request message**3 **Purpose:** Find keys by attributes known to the client (e.g. data set).4 **Required parameters:**

5 — At least one property that the client wants to match against the available keys.

6 **Results and response:** Return all matching keys, formatted as an array.

- 1 **Example:** A query for a key of a particular type, requested by a particular user, and used to encrypt one or
 2 more tapes within a given pool.

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:km="http://www.IEEE.org/1619/KeyManagement/">

  <soap:Body>

    <km:KeyManagement_QueryKey>

      <kex_input>

        <keyExchangeStructure>

          <keyObject>

            <keyID>
              <keyClassname>kcn.2007-05.org.IEEE.1619:XTS-AES-256
            </keyClassname>
            </keyID>

            <keyOrigin>
              <requestorID>user1@example.com</requestorID>
            </keyOrigin>

          </keyObject>

          <dataSetBindings>

            <containerAttributes>
              <storagePoolLabel>Ireland</storagePoolLabel>
            </containerAttributes>

          </dataSetBindings>

        </keyExchangeStructure>

      </kex_input>

    </km:KeyManagement_QueryKey>

  </soap:Body>

</soap:Envelope>

```

3

- 1 **Response:** The KM Server responds with two key exchange structures that match the criteria. Only one of
 2 which is no longer assigned to the current pool.

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:km="http://www.IEEE.org/1619/KeyManagement/">

  <soap:Body>

    <km:KeyManagement_QueryKeyResponse>

      <kex_output>

        <keyExchangeStructure>

          <keyObject>

            <keyID>
              <keyClassname>kcn.2007-05.org.IEEE.1619:XTS-AES-256
              </keyClassName>
              <keyFormat>kfn.2007-05.org.IEEE.1619:generic</keyFormat>
              <keyIdSpace>example.com</keyIdSpace>
              <KeyIdValue>987654</keyIdValue>
            </keyID>

            <keyContents>
              <secretContents>SecretBinaryValue</secretContents>
            </keyContents>

            <keyOrigin>
              <generationTime>June 1, 2000</generationTime>
              <originatorID>kms2@example.com</originatorID>
              <requestorID>user1@example.com</requestorID>
            </keyOrigin>

          </keyObject>

          <dataSetBindings>

            <containerAttributes>
              <storagePoolLabel>Ireland</storagePoolLabel>
              <storageTapeLabel>0029</storageTapeLabel>
            </containerAttributes>

          </dataSetBindings>

          <keyAssignmentPolicy>
            <attribute name="storagePoolLabel" value="Spain">
          </keyAssignmentPolicy>

          <replicationPolicy>
            <keyManagerID>kms5@example.com</keyManagerID>
            <commitment_request>notify</commitment_request>
            <Delay>20</Delay>
          </replicationPolicy>

          <retentionPolicy>
            <expirationTime>January 1, 2020</expirationTime>
            <disableTime>January 1, 2010</disableTime>
            <purgeTime>January 1,2021</purgeTime>
          </retentionPolicy>

        </keyExchangeStructure>
  
```

```

</kex_output>

<kex_output>

  <keyExchangeStructure>

    <keyObject>

      <keyID>
        <keyClassName>kcn.2007-05.org.IEEE.1619:XTS-AES-256
        </keyClassName>
        <keyFormat>kfn.2007-05.org.IEEE.1619:generic</keyFormat>
        <keyIdSpace>example.com</keyIdSpace>
        <KeyIdValue>123456</keyIdValue>
      </keyID>

      <keyContents>
        <secretContents>SecretBinaryValue</secretContents>
      </keyContents>

      <keyOrigin>
        <generationTime>January 1, 2000</generationTime>
        <originatorID>kms1@example.com</originatorID>
        <requestorID>user1@example.com</requestorID>
      </keyOrigin>

    </keyObject>

    <dataSetBindings>

      <containerAttributes>
        <storagePoolLabel>Ireland</storagePoolLabel>
        <storageTapeLabel>0002</storageTapeLabel>
      </containerAttributes>

      <dataAttributes>
        <attribute name="customer data">
      </dataAttributes>

    </dataSetBindings>

    <keyAssignmentPolicy>
      <attribute name="storagePoolLabel" value="Ireland">
    </keyAssignmentPolicy>

    <replicationPolicy>
      <keyManagerID>kms2@example.com</keyManagerID>
      <commitment_request>notify</commitment_request>
      <Delay>20</Delay>
    </replicationPolicy>

    <retentionPolicy>
      <expirationTime>January 1, 2020</expirationTime>
      <disableTime>January 1, 2010</disableTime>
      <purgeTime>January 1,2021</purgeTime>
    </retentionPolicy>

  </keyExchangeStructure>

</kex_output>

</km:KeyManagement_QueryKeyResponse>

```

```

    </soap:Body>
</soap:Envelope>

```

1 9.1.5 storeKey request message

2 **Purpose:** The client supplies a subset of the data in a key exchange structure. This command may not be
3 used to update key policies, but only to store new keys.

4 **Required parameters:**

5 —Key Object parameter

6 **Optional parameters:**

7 —All of the key policy elements may be sent

8 **Result and response:** The key shall be inserted into the KM Server with the supplied policies. If the key
9 already exists in the KMS, the command shall fail.

10 **NOTE—** Parameter checking shall be performed by the KM Server and appropriate error returned for
11 malformed requests.

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:km="http://www.IEEE.org/1619/KeyManagement/">

  <soap:Body>

    <km:KeyManagement_storeKey>

      <kex_input>

        <keyExchangeStructure>

          <keyObject>

            <keyID>
              <keyClassname>kcn.2007-05.org.IEEE.1619:XTS-AES-256
            </keyClassname>
              <keyFormat>kfn.2007-05.org.IEEE.1619:generic</keyFormat>
              <keyIdSpace>example.com</keyIdSpace>
              <KeyIdValue>123456<KeyIdValue>
            </keyID>

            <keyContents>
              <secretContents>SecretBinaryValue</secretContents>
            </keyContents>

            <keyOrigin>
              <generationTime>January 1, 2000</generationTime>
              <originatorID>kms1@example.com</originatorID>
              <requestorID>user1@example.com</requestorID>
            </keyOrigin>

          </keyObject>

          <dataSetBindings>

            <containerAttributes>

```

```

        <storagePoolLabel>Ireland</storagePoolLabel>
        <storageTapeLabel>0002</storageTapeLabel>
    </containerAttributes>

    <dataAttributes>
        <attribute name="customer data">
    </dataAttributes>

</dataSetBindings>

<keyAssignmentPolicy>
    <attribute name="storagePoolLabel" value="Ireland">
</keyAssignmentPolicy>

<replicationPolicy>
    <keyManagerID>kms2@example.com</keyManagerID>
    <commitment_request>notify</commitment_request>
    <Delay>20</Delay>
</replicationPolicy>

<retentionPolicy>
    <expirationTime>January 1, 2020</expirationTime>
    <disableTime>January 1, 2010</disableTime>
    <purgeTime>January 1,2021</purgeTime>
</retentionPolicy>

    </keyExchangeStructure>

    </kex_input>

    </km:KeyManagement_storeKey>

</soap:Body>
</soap:Envelope>

```

1

2 **Return parameters:** The success message is returned on successful key store.3 **NOTE**—this method shall fail if the key is already in the KMS.

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:km="http://www.IEEE.org/1619/KeyManagement/">

    <soap:Body>

        <km:KeyManagement_storeKey>

            <kex_output>
                <parameter name="kex_status">OK</parameter>
            </kex_output>

        </km:KeyManagement_storeKey>

    </soap:Body>

</soap:Envelope>

```

4 **9.1.6 updateKeyInfo request message**

1 **Purpose:** To update a key exchange structure already in the KMS. This command may not be used to store
2 new keys.

3 **Required parameters:**

4 — Key ID component of the Key Object parameter

5 **Optional parameters:**

6 — All of the key policy elements may be sent, with the exception of the key contents parameter

7 **Result and response:**

8 If the key ID is already in the KMS, then the parameters assigned to the key ID shall be updated.

9 NOTE—Parameter checking shall be performed by the KM Server and appropriate error returned for malformed
10 requests. Caution should be taken when updating data set bindings.

11 9.1.7 generateKey request message

12 **Purpose:** The client requests that the KM Server generate a key with the supplied key format and
13 classname parameters, and the KM Server returns the key ID parameter of the generated key. This
14 command allows one client to generate a key on behalf of another entity.

15 **Required parameters:**

16 — Key Classname and Key Format fields.

17 **Optional parameters:**

18 — All of the key policy elements may be sent, with the exception of the key contents element,
19 generation time, and the Key ID value.

20 **Result and response:**

21 A key with the supplied attributes is generated by the KM Server and the Key ID (but not the key) is
22 returned. If the policy settings may not be sent, the entire command shall fail.

23 **Example:**

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:km="http://www.IEEE.org/1619/KeyManagement/">

  <soap:Body>

    <km:KeyManagement_generateKey>

      <kex_input>
        <keyClassname>kcn.2007-05.org.IEEE.1619:XTS-AES-256</keyClassName>
        <keyFormat>kfn.2007-05.org.IEEE.1619:generic</keyFormat>
      </kex_input>

    </km:KeyManagement_generateKey>

  </soap:Body>

</soap:Envelope>

```

24

25 **Return params:** The key ID is returned on success.

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:km="http://www.IEEE.org/1619/KeyManagement/">

  <soap:Body>

    <km:KeyManagement_generateKeyResponse>

      <kex_output>

        <keyID>
          <keyClassName>kcn.2007-05.org.IEEE.1619:XTS-AES-256
          </keyClassName>
          <keyFormat>kfn.2007-05.org.IEEE.1619:generic</keyFormat>
          <keyIdSpace>example.com</keyIdSpace>
          <keyIdValue>123456</keyIdValue>
        </keyID>

      </kex_output>

    </km:KeyManagement_generateKeyResponse >

  </soap:Body>

</soap:Envelope>

```

1

2 9.1.8 getSeed request message

3 **Purpose:** The client requests 64 bytes of entropy.

4 **Required Parameters:**

5 — none

6 **Optional Parameters:**

7 — none

8 **Result and response:**

9 64 bytes of entropy are provided to the client, in the entropy format.

10 **Prerequisites:**

11 Client should have been authenticated with KM Server prior to the request.

12 **Example:** Request for seed material.

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:km="http://www.IEEE.org/1619/KeyManagement/">

  <soap:Body>
    <km:KeyManagement_getSeed>
    </km:KeyManagement_getSeed>
  </soap:Body>

</soap:Envelope>

```

13

14 **Return params:** KM Server returns entropy.

```

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:km="http://www.IEEE.org/1619/KeyManagement/">

  <soap:Body>

    <km:KeyManagement_getSeedResponse>

      <kex_output>
        <keyClassname>kcn.2007.05.org.IEEE.1619:entropy</keyClassname>
        <keyFormat>kcn.2007-05.org.IEEE.1619:generic</keyFormat>
        <secretContents>SeedContents</secretContents>
      </kex_output>

    </km:KeyManagement_getSeedResponse>

  </soap:Body>

</soap:Envelope>

```

1

2 **9.2 Binary command based protocol**

3 This section describes a lightweight binary protocol between the KM Client and KM Server. This protocol
4 supports the above mentioned commands.

5 **9.2.1 Overview**

6 *[Content TBD]*

7 **9.2.2 Get key by storage attribute**

8 *[Content TBD]*

9 **9.2.3 Get key by ID**

10 *[Content TBD]*

11 **9.2.4 Query key**

12 *[Content TBD]*

13 **9.2.5 Store key**

14 *[Content TBD]*

15 **9.2.6 UpdateKey info**

16 *[Content TBD]*

17 **9.2.7 Generate key**

18 *[Content TBD]*

19 **9.2.8 GetSeed**

20 *[Content TBD]*

1 **10. Key Management Transport**

2 *[Content TBD]*

3

1 **Annex A**

2 (normative)

3 **Minimum Requirements to Meet P1619.3**

4 **A.1 KM Client Requirements**

5 The following information provides a minimum set of requirements to be met by a KM Client in order to be
6 considered compliant with the P1619.3 Standard.

7 **A.1.1 Name Space Requirements**

8

9 **A.1.2 Required Objects**

10

11 **A.1.3 Required Operations**

12

13 **A.1.4 Required Messaging**

14

15 **A.1.5 Transport Requirements**

16

17 **A.2 KM Server Requirements**

18 The following information provides a minimum set of requirements to be met by a KM Server in order to
19 be considered compliant with the P1619.3 Standard.

20 **A.2.1 Name Space Requirements**

21

22 **A.2.2 Required Objects**

23

24 **A.2.3 Required Policy**

25

26 **A.2.4 Required Operations**

27

1 **A.2.5 Required Messaging**

2

3 **A.2.6 Transport Requirements**

4

1 **Annex B**

2 (informative)

3 **Bibliography**

4 [B1] ISO/IEC 18033-1:2005 Information technology - Security techniques - Encryption algorithms - Part
5 1: General

6 [B2] ISO/IEC 18033-2 Information technology - Security techniques - Encryption algorithms - Part 2:
7 Asymmetric ciphers

8 [B3] IEEE 1363-2000 Standard Specifications for Public Key Cryptography

9 [B4] IEEE Std 1363a-2004 IEEE Standard Specifications for Public-Key Cryptography - Amendment 1:
10 Additional Techniques

11 [B5] NIST Special Publication 800-56A Recommendation for Pair-Wise Key Establishment Schemes
12 Using Discrete Logarithm Cryptography

13 [B6] ASN X9.24 – Retail Financial Services Symmetric Key Management – Part 1: Using Symmetric
14 Techniques.

15 [B7] ASN X9.24 – Retail Financial Services Symmetric Key Management – Part 2: Using Asymmetric
16 Techniques for the Distribution of Symmetric Keys

17 [B8] IEEE 100 – The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition

18 In each of the following examples, a client type is identified, followed by example encryption policies that
19 may be appropriate in that environment, as well as security goals that the encryption is meant to achieve.

1 **Annex C**

2 (informative)

3 **Example Use Cases**

4 **C.1 Tape libraries with encrypting drives**

5 The following illustrates the sequence of events between a tape library that is managing tape drives, tape
6 volumes and is responsible to communicate with a key manager for obtaining and using the key for
7 encryption and subsequent data access.

8 For encryption, the library will query the KM Server for the appropriate key, given the data attributes. For
9 decryption, the tape drive will write the key ID onto the tape header during encryption, and will request the
10 key by ID during decryption.

11 — KM Server one time setup operations

12 1) Client Registration – the KMS administrator enrolls the library as a KM Client.

13 2) Policy setup:

14 i) Administrator defines an encryption policy:

15 `Key_Per_Tape`

16 `{`

17 `Encryption Mode: GCM-128-AES-256`

18 `Key Generation Mode: One_Per_Tape`

19 `}`

20 ii) Administrator defines a retention policy:

21 `7_Year_Policy`

22 `{`

23 `Expiration Time: 30 days after creation`

24 `Disable Time: 7 Years, 0 Months, 0 Days`

25 `Purge Time: 30 days after disable`

26 `}`

27 iii) Administrator defines a key assignment policy:

28 `Finance_Data_Backup_DataSet`

29 `{`

30 `Encryption Data Type: Tape Tape Cartridge`

31 `Label Range: E0001 – E0010`

```

1           Usage_Model: Key_Per_Tape
2           Retention Policy: 7_Year_Policy
3           }
4  — Library client operations
5     3) Writing to new media
6         i) Library requests encryption key: GetKeyDataAttribute(Attrib:TapeVolumeId = E0005)
7         ii) KM Server matches the TapeVolumeID to Finance_Data_Backup_DataSet. KM Server
8             generates a GCM-128-AES-256 key and applies both encryption and retentions policies
9             to the newly created key KEY001 and returns it to the client.
10        iii) Library uses KEY001 to encrypt volume E0005.
11     4) Reading from media
12        i) Library loads volume and retrieves KeyID KEY001
13        ii) Library calls GetKeyById(KEY001)
14        iii) KM Server checks ACLs and retention policy. If checks pass, KM Server sends the key
15             to the client.
16        iv) Library uses KEY001 to decrypt the information stored in the volume

```

17 C.2 Backup applications

18 The backup application may support complex data management policies, such as data set definitions,
 19 retention policies and distribution polices. Adding encryption to this environment adds strong enforcement
 20 of these policies.

21 A data set may consist of directories, files, shares, or any other abstraction as supported by the backup
 22 application.

23 In this case, the backup application will send the data set identifiers to the KM Server, for key generation
 24 and forward the KeyID to the encryption environment for retrieval of the actual key to be used in
 25 encryption.

26 C.3 Laptop/desktop encryption

27 In this case, there may be a single key assigned to a partition, or to a set of physical disks. Special sectors
 28 may be designated that are encrypted with separate keys. In this case, to laptop will request a key by data
 29 attributes (e.g. “special sector” or “user partition”) and the KM Server will provide the appropriate key. The
 30 key will be stored in the device, and will be made available for purposes of emergency recovery should the
 31 client node be damaged or unavailable.

32 C.4 NAS filer encryption

33 In this case, data sets may correspond to shares, mount points, directories, or files, or possibly more
 34 granular data sets. The end point will request keys based on the data set attributes as in the other use cases.

35

36

1 Annex D

2 (informative)

3 XML Schema Definitions

```

<!-- Note: minOccurs default is 1 -->

<wsdl:definitions name="P1619-3Service"
targetNamespace="http://siswg.net"
xmlns:P1619-3="http://siswg.net"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

  <!-- Type definitions -->

  <wsdl:types>

    <xsd:schema targetNamespace="http://siswg.net">

      <xsd:element name="SO_GUID" type="xsd:string"/>
      <xsd:element name="OID" type="xsd:string"/>
      <xsd:element name="UTC" type="xsd:dateTime"/>

      <xsd:complexType name="KeyAssignmentPolicy">
        <xsd:sequence>
          <xsd:element name="KEY_ASSIGNMENT_POLICY_ID"
            type="xsd:string"/>
          <xsd:element name="DESCRIPTION" type="xsd:string"/>
          <xsd:element name="REALM_ASSOCIATIONS" type="xsd:string"
            maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="RetentionPolicy">
        <xsd:sequence>
          <xsd:element name="RETENTION_POLICY_ID" type="xsd:string"/>
          <xsd:element name="DESCRIPTION" type="xsd:string"/>
          <xsd:element name="REALM_ASSOCIATIONS" type="xsd:string"
            maxOccurs="unbounded"/>
          <xsd:element name="T_EXPIRATION" type="P1619-3:UTC"/>
          <xsd:element name="T_DISABLE" type="P1619-3:UTC"/>
        </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="AccessPolicy">
        <xsd:sequence>
          <xsd:element name="SO_GUID" type="P1619-3:SO_GUID"/>
          <xsd:element name="CryptographicUnitID" type="P1619-3:SO_GUID"
            maxOccurs="unbounded"/>
          <xsd:element name="REALM_ASSOCIATIONS" type="xsd:string"
            maxOccurs="unbounded"/>
        </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="CachingPolicy">
        <xsd:sequence>

```

```

        <xsd:element name="Type" type="xsd:string"/>
        <xsd:element name="Interval" type="xsd:duration"/>
        <xsd:element name="HARDWARE_TYPE" type="xsd:string"/>
        <xsd:element name="DESTINATION" type="xsd:string"/>
        <xsd:element name="STANDARDS_CONFORMANCE" type="xsd:string"/>
        <xsd:element name="CACHE_TIME" type="xsd:duration"/>
        <xsd:element name="USAGE_COUNT" type="xsd:unsignedInt"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="WrappingPolicy">
    <xsd:sequence>
        <xsd:element name="Type" type="xsd:string"/>
        <xsd:element name="Mode" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Key">
    <xsd:sequence>
        <xsd:element name="KeyID" type="P1619-3:SO_GUID"/>
        <xsd:element name="FriendlyName" type="xsd:string"
            minOccurs="0"/>
        <xsd:element name="State" type="xsd:string"/>
        <xsd:element name="CipherType" type="P1619-3:OID"/>
        <xsd:element name="Version" type="xsd:unsignedInt"/>
        <xsd:element name="EditDateTime" type="P1619-3:UTC"/>
        <xsd:element name="TimeExpired" type="P1619-3:UTC"/>
        <xsd:element name="TimeDisabled" type="P1619-3:UTC"/>
        <xsd:element name="TimeCached" type="xsd:duration"/>
        <xsd:element name="Description" type="xsd:string"/>
        <xsd:element name="KeyBlob">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="ProtocolVersion"
                        type="xsd:unsignedInt"/>
                    <xsd:element name="WrappingType" type="xsd:string"/>
                    <xsd:element name="Length" type="xsd:unsignedInt"/>
                    <xsd:element name="Data" type="xsd:hexBinary"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="CachingPolicy"
            type="P1619-3:CachingPolicy"/>
        <xsd:element name="WrappingPolicy"
            type="P1619-3:WrappingPolicy"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="KeyTemplate">
    <xsd:sequence>
        <xsd:element name="KeyTemplateID" type="P1619-3:SO_GUID"/>
        <xsd:element name="FriendlyName" type="xsd:string"
            minOccurs="0"/>
        <xsd:element name="CipherType" type="P1619-3:OID"/>
        <xsd:element name="Version" type="xsd:unsignedInt"/>
        <xsd:element name="EditDateTime" type="P1619-3:UTC"/>
        <xsd:element name="Description" type="xsd:string"/>
        <xsd:element name="CachingPolicy"
            type="P1619-3:CachingPolicy"/>
        <xsd:element name="WrappingPolicy"
            type="P1619-3:WrappingPolicy"/>
    </xsd:sequence>
</xsd:complexType>

```

```

<xsd:complexType name="EndPointType">
  <xsd:sequence>
    <xsd:element name="EndPointTypeID" type="xsd:string"/>
    <xsd:complexType name="Capabilities">
      <xsd:element name="Declaration" type="xsd:string"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:complexType>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Realm">
  <xsd:sequence>
    <xsd:element name="RealmID" type="xsd:string"/>
    <xsd:element name="Description" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="CryptographicUnit">
  <xsd:sequence>
    <xsd:element name="CryptographicUnitID"
      type="P1619-3:SO_GUID"/>
    <xsd:element name="EndPointType" type="P1619-3:EndPointType"/>
    <xsd:element name="WrappingPolicy"
      type="P1619-3:WrappingPolicy"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="KM_Client">
  <xsd:sequence>
    <xsd:element name="CredentialType" type="xsd:string"/>
    <xsd:element name="EndPointType" type="P1619-3:EndPointType"/>
    <xsd:element name="WrappingPolicy"
      type="P1619-3:WrappingPolicy"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="DataSet">
  <xsd:sequence>
    <xsd:element name="Name" type="xsd:string"/>
    <xsd:element name="Value" type="xsd:hexBinary"/>
    <xsd:element name="ValueSize" type="xsd:unsignedInt"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ClientGroup">
  <xsd:sequence>
    <xsd:element name="Type" type="xsd:string"/>
    <xsd:element name="CLIENT_SO_GUID" type="P1619-3:SO_GUID"
      maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="KeyGroup">
  <xsd:sequence>
    <xsd:element name="Type" type="xsd:string"/>
    <xsd:element name="CLIENT_SO_GUID" type="P1619-3:SO_GUID"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="RegisterClient">
  <xsd:complexType>
    <xsd:sequence>

```

```

        <xsd:element name="KM_Client" type="P1619-3:KM_Client" />
    </xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="RegisterClientResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="ReturnCode" type="xsd:int" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="Authenticate">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="KM_Client" type="P1619-3:KM_Client" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="AuthenticateResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="ReturnCode" type="xsd:int" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="CapabilityNegotiation">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="KM_Client" type="P1619-3:KM_Client" />
            <xsd:element name="Capabilities">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="Declaration" type="xsd:string"
                            minOccurs="0" maxOccurs="unbounded" />
                        <xsd:element name="Question" type="xsd:string"
                            minOccurs="0" maxOccurs="unbounded" />
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="CapabilityNegotiationResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Capabilities">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="Answer" type="xsd:string"
                            minOccurs="0" maxOccurs="unbounded" />
                        <xsd:element name="QuestionUnknown" type="xsd:string"
                            minOccurs="0" maxOccurs="unbounded" />
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

```

```

<xsd:element name="GetServerCapabilities">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="KM_Client" type="P1619-3:KM_Client"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="GetServerCapabilitiesResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Capabilities">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Declaration" type="xsd:string"
              minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="PushAuditMessage">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="KM_Client" type="P1619-3:KM_Client"/>
      <xsd:element name="KeyID" type="P1619-3:SO_GUID"/>
      <xsd:element name="Message" type="xsd:string"
        minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="PushAuditMessageResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="ReturnCode" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="GetRandomBytes">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="KM_Client" type="P1619-3:KM_Client"/>
      <xsd:element name="BytesNumber" type="xsd:unsignedInt"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="GetRandomBytesResponse">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Bytes" type="xsd:hexBinary"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="KeyCreate">
  <xsd:complexType>
    <xsd:sequence>

```

```

        <xsd:element name="KM_Client" type="P1619-3:KM_Client"/>
        <xsd:element name="CryptographicUnit"
            type="P1619-3:CryptographicUnit"/>
        <xsd:element name="Key" type="P1619-3:Key" minOccurs="0"/>
        <xsd:element name="KeyTemplate" type="P1619-3:KeyTemplate"
            minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>

<xsd:element name="KeyCreateResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="ReturnCode" type="xsd:int"/>
            <xsd:element name="Key" type="P1619-3:Key" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="KeyGet">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="KM_Client" type="P1619-3:KM_Client"/>
            <xsd:element name="CryptographicUnit"
                type="P1619-3:CryptographicUnit"/>
            <xsd:element name="Key" type="P1619-3:Key"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="KeyGetResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="ReturnCode" type="xsd:int"/>
            <xsd:element name="Key" type="P1619-3:Key" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="KeyStore">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="KM_Client" type="P1619-3:KM_Client"/>
            <xsd:element name="CryptographicUnit"
                type="P1619-3:CryptographicUnit"/>
            <xsd:element name="Key" type="P1619-3:Key"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name="KeyStoreResponse">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="ReturnCode" type="xsd:int"/>
            <xsd:element name="KeyID" type="P1619-3:SO_GUID"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

</xsd:schema>

</wsdl:types>

```

```

<!-- Message definitions -->

<wsdl:message name="RegisterClient">
  <wsdl:part name="parameters" element="P1619-3:RegisterClient"/>
</wsdl:message>

<wsdl:message name="RegisterClientResponse">
  <wsdl:part name="parameters" element="P1619-3:RegisterClientResponse"/>
</wsdl:message>

<wsdl:message name="Authenticate">
  <wsdl:part name="parameters" element="P1619-3:Authenticate"/>
</wsdl:message>

<wsdl:message name="AuthenticateResponse">
  <wsdl:part name="parameters" element="P1619-3:AuthenticateResponse"/>
</wsdl:message>

<wsdl:message name="CapabilityNegotiation">
  <wsdl:part name="parameters" element="P1619-3:CapabilityNegotiation"/>
</wsdl:message>

<wsdl:message name="CapabilityNegotiationResponse">
  <wsdl:part name="parameters"
    element="P1619-3:CapabilityNegotiationResponse"/>
</wsdl:message>

<wsdl:message name="GetServerCapabilities">
  <wsdl:part name="parameters" element="P1619-3:GetServerCapabilities"/>
</wsdl:message>

<wsdl:message name="GetServerCapabilitiesResponse">
  <wsdl:part name="parameters"
    element="P1619-3:GetServerCapabilitiesResponse"/>
</wsdl:message>

<wsdl:message name="PushAuditMessage">
  <wsdl:part name="parameters" element="P1619-3:PushAuditMessage"/>
</wsdl:message>

<wsdl:message name="PushAuditMessageResponse">
  <wsdl:part name="parameters" element="P1619-3:PushAuditMessageResponse"/>
</wsdl:message>

<wsdl:message name="GetRandomBytes">
  <wsdl:part name="parameters" element="P1619-3:GetRandomBytes"/>
</wsdl:message>

<wsdl:message name="GetRandomBytesResponse">
  <wsdl:part name="parameters" element="P1619-3:GetRandomBytesResponse"/>
</wsdl:message>

<wsdl:message name="KeyCreate">
  <wsdl:part name="parameters" element="P1619-3:KeyCreate"/>
</wsdl:message>

<wsdl:message name="KeyCreateResponse">
  <wsdl:part name="parameters" element="P1619-3:KeyCreateResponse"/>
</wsdl:message>

<wsdl:message name="KeyGet">
  <wsdl:part name="parameters" element="P1619-3:KeyGet"/>
</wsdl:message>

```

```

<wsdl:message name="KeyGetResponse">
  <wsdl:part name="parameters" element="P1619-3:KeyGetResponse" />
</wsdl:message>

<wsdl:message name="KeyStore">
  <wsdl:part name="parameters" element="P1619-3:KeyStore" />
</wsdl:message>

<wsdl:message name="KeyStoreResponse">
  <wsdl:part name="parameters" element="P1619-3:KeyStoreResponse" />
</wsdl:message>

<!-- Port type definitions -->

<wsdl:portType name="P1619-3">

  <wsdl:operation name="RegisterClient">
    <wsdl:input message="RegisterClient" />
    <wsdl:output message="RegisterClientResponse" />
  </wsdl:operation>

  <wsdl:operation name="Authenticate">
    <wsdl:input message="Authenticate" />
    <wsdl:output message="AuthenticateResponse" />
  </wsdl:operation>

  <wsdl:operation name="CapabilityNegotiation">
    <wsdl:input message="CapabilityNegotiation" />
    <wsdl:output message="CapabilityNegotiationResponse" />
  </wsdl:operation>

  <wsdl:operation name="GetServerCapabilities">
    <wsdl:input message="GetServerCapabilities" />
    <wsdl:output message="GetServerCapabilitiesResponse" />
  </wsdl:operation>

  <wsdl:operation name="PushAuditMessage">
    <wsdl:input message="PushAuditMessage" />
    <wsdl:output message="PushAuditMessageResponse" />
  </wsdl:operation>

  <wsdl:operation name="GetRandomBytes">
    <wsdl:input message="GetRandomBytes" />
    <wsdl:output message="GetRandomBytesResponse" />
  </wsdl:operation>

  <wsdl:operation name="KeyCreate">
    <wsdl:input message="KeyCreate" />
    <wsdl:output message="KeyCreateResponse" />
  </wsdl:operation>

  <wsdl:operation name="KeyGet">
    <wsdl:input message="KeyGet" />
    <wsdl:output message="KeyGetResponse" />
  </wsdl:operation>

  <wsdl:operation name="KeyStore">
    <wsdl:input message="KeyStore" />
    <wsdl:output message="KeyStoreResponse" />
  </wsdl:operation>

</wsdl:portType>

```

```

<!-- Binding definitions -->

<wsdl:binding name="P1619-3PortBinding" type="P1619-3:P1619-3">

  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>

  <wsdl:operation name="RegisterClient">
    <soap:operation soapAction=""/>
    <wsdl:input><soap:body use="literal"/></wsdl:input>
    <wsdl:output><soap:body use="literal"/></wsdl:output>
  </wsdl:operation>

  <wsdl:operation name="Authenticate">
    <soap:operation soapAction=""/>
    <wsdl:input><soap:body use="literal"/></wsdl:input>
    <wsdl:output><soap:body use="literal"/></wsdl:output>
  </wsdl:operation>

  <wsdl:operation name="CapabilityNegotiation">
    <soap:operation soapAction=""/>
    <wsdl:input><soap:body use="literal"/></wsdl:input>
    <wsdl:output><soap:body use="literal"/></wsdl:output>
  </wsdl:operation>

  <wsdl:operation name="GetServerCapabilities">
    <soap:operation soapAction=""/>
    <wsdl:input><soap:body use="literal"/></wsdl:input>
    <wsdl:output><soap:body use="literal"/></wsdl:output>
  </wsdl:operation>

  <wsdl:operation name="PushAuditMessage">
    <soap:operation soapAction=""/>
    <wsdl:input><soap:body use="literal"/></wsdl:input>
    <wsdl:output><soap:body use="literal"/></wsdl:output>
  </wsdl:operation>

  <wsdl:operation name="GetRandomBytes">
    <soap:operation soapAction=""/>
    <wsdl:input><soap:body use="literal"/></wsdl:input>
    <wsdl:output><soap:body use="literal"/></wsdl:output>
  </wsdl:operation>

  <wsdl:operation name="KeyCreate">
    <soap:operation soapAction=""/>
    <wsdl:input><soap:body use="literal"/></wsdl:input>
    <wsdl:output><soap:body use="literal"/></wsdl:output>
  </wsdl:operation>

  <wsdl:operation name="KeyGet">
    <soap:operation soapAction=""/>
    <wsdl:input><soap:body use="literal"/></wsdl:input>
    <wsdl:output><soap:body use="literal"/></wsdl:output>
  </wsdl:operation>

  <wsdl:operation name="KeyStore">
    <soap:operation soapAction=""/>
    <wsdl:input><soap:body use="literal"/></wsdl:input>
    <wsdl:output><soap:body use="literal"/></wsdl:output>
  </wsdl:operation>

</wsdl:binding>

```

```
<wsdl:service name="P1619-3Service">  
  <wsdl:port name="P1619-3" binding="P1619-3:P1619-3PortBinding">  
    <soap:address location="http://your_km_ip:%d%s" />  
  </wsdl:port>  
</wsdl:service>  
  
</wsdl:definitions>
```

1

1 **Annex E**

2 (informative)

3 **Comparison of P1619.3 Key Lifecycle Model with Other Standards**

4 **E.1 Key State Comparisons**

5 Table D.1 compares the mappings from the P1619.3 key states to the key states defined in the NIST SP
6 800-57 lifecycle model and the IEC/ISO 11770-1 lifecycle model:

7 **Table 6 Standard Key State Comparisons**

P1619.3 Key State	NIST SP 800-57 Key State	IEC/ISO 11770-1 Key State	Notes
Pre-Activation	Pre-activation	Pending Active	Notes 1 & 2
Protect-and-Process	Active	Active	Notes 3 & 4
Process-Only	Active	Active	Notes 3 & 4
Expired	Deactivated	Deactivated	Notes 3 & 4
Disabled	Deactivated	Deactivated	Notes 3 & 4
Compromised	Compromised	None	Notes 3 & 5
Disabled-Compromised	Compromised	None	Notes 3 & 5
Destroyed	Destroyed	Destroyed (Implied)	Notes 1 & 6
Destroyed-Compromised	Destroyed Compromised	None	Notes 1 & 5
Purged	None	None	Notes 7 & 5

8 Note 1: IEEE P1619.3 key state is identical to the key state defined in the NIST SP 800-57 standard.

9 Note 2: IEEE P1619.3 key state is identical to the key state defined in the IEC/ISO 11770-1 standard.

10 Note 3: IEEE P1619.3 key state is a substate of the key state defined in the NIST SP 800-57 standard.

11 Note 4: IEEE P1619.3 key state is a substate of the key state defined in the IEC/ISO 11770-1
12 standard.

13 Note 5: IEEE P1619.3 key state is a new state not defined in the IEC/ISO 11770-1 standard.

14 Note 6: IEEE P1619.3 key state is the same as the state implied in the IEC/ISO 11770-1 standard.

15 Note 7: IEEE P1619.3 key state is a new state not defined in the NIST SP 800-57 standard.

16 **E.2 Key Transition Comparisons**

17 Table D.2 compares the mappings from the P1619.3 key transitions to the key transitions defined in the
18 NIST SP 800-57 lifecycle model and the IEC/ISO 11770-1 lifecycle model:

1

Table 7 Standard Key Transition Comparisons

P1619.3 Key Transition	NIST SP 800-57 Key Transition	IEC/ISO 11770-1 Key Transition	Notes
Create	Transition 1	Generation	Notes 1 & 2
Destroy	Transition 2	Destruction	Notes 1 & 2
Activate	Transition 4	Activation	Notes 1 & 2
Compromise	Transition 3	None	Notes 1 & 4
Process-Only	None	None	Notes 3 & 4
Compromise	Transition 5	None	Notes 1 & 4
Expire	Transition 6	Deactivation	Notes 1 & 2
Compromise	Transition 5	None	Notes 1 & 4
Disable	None	None	Notes 3 & 4
Compromise	Transition 8	None	Notes 1 & 4
11) Disable	None	None	Notes 3 & 4
12) Destroy	Transition 7	Destruction	Notes 1 & 2
13) Compromise	Transition 8	None	Note 1 & 4
14) Recover	None	Reactivation (see note)	Notes 5 & 6
15) Destroy	Transition 9	Destruction	Notes 1 & 2
16) Recover	None	Reactivation (see note)	Notes 5 & 6
17) Purge	None	None	Notes 3 & 4
18) Compromise	Transition 10	None	Note 1 & 4
19) Purge	None	None	Note 3 & 4

2

3 Note 1: IEEE P1619.3 key state transition is identical to the key state transition defined in the NIST
4 SP 800-57 standard.

5 Note 2: IEEE P1619.3 key state transition is identical to the key state transition defined in the
6 IEC/ISO 11770-1 standard.

7 Note 3: IEEE P1619.3 key state transition is a new key state transition not defined in the NIST SP
8 800-57 standard.

9 Note 4: IEEE P1619.3 key state transition is a new key state transition not defined in the IEC/ISO
10 11770-1 standard.

11 Note 5: IEEE P1619.3 key state transition is implied by the discussion in the NIST SP 800-57
12 standard, but is not explicitly shown.

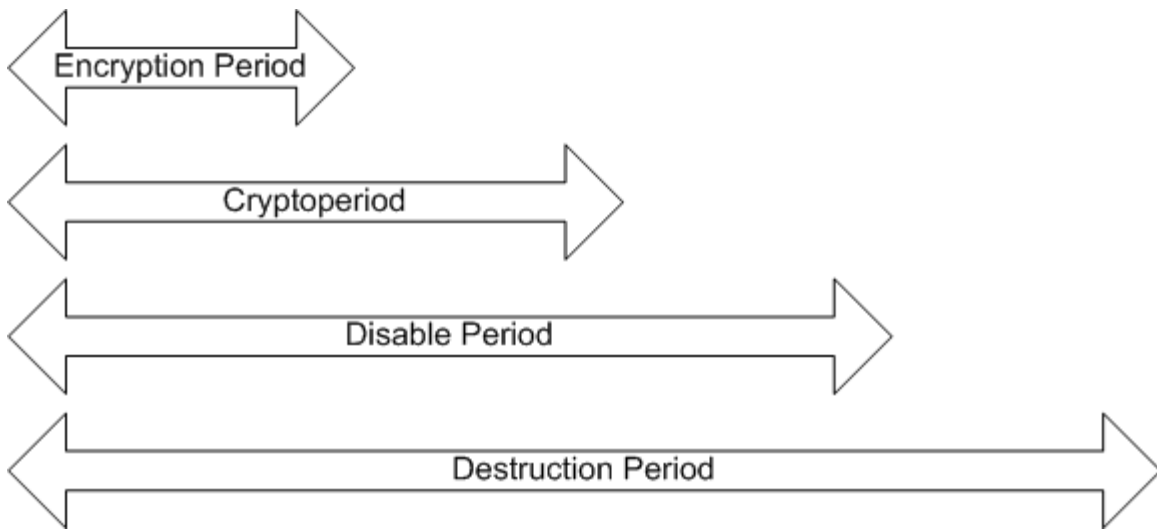
13 Note 6: IEEE P1619.3 key state transition is similar to the Reactivation transition described in
14 IEC/ISO 11770-1 standard, but does not result in the key transition back to the active state.

15 **E.3 Key Time Period Comparisons**

16 The P1619.3 key life cycle time periods are based on the time periods defined in NIST SP 800-57;
17 however, this standard renames the time periods, places limits on their relationships, and adds two
18 additional time periods. These changes were necessary to support the concept of long term key usage
19 required for encryption of stored data applications.

1 Figure D.1 illustrates the key related time periods defined in the P1619.3 standard. Figure D.2 provides a
 2 visual depiction of the key related time periods defined in NIST SP 800-57.

3



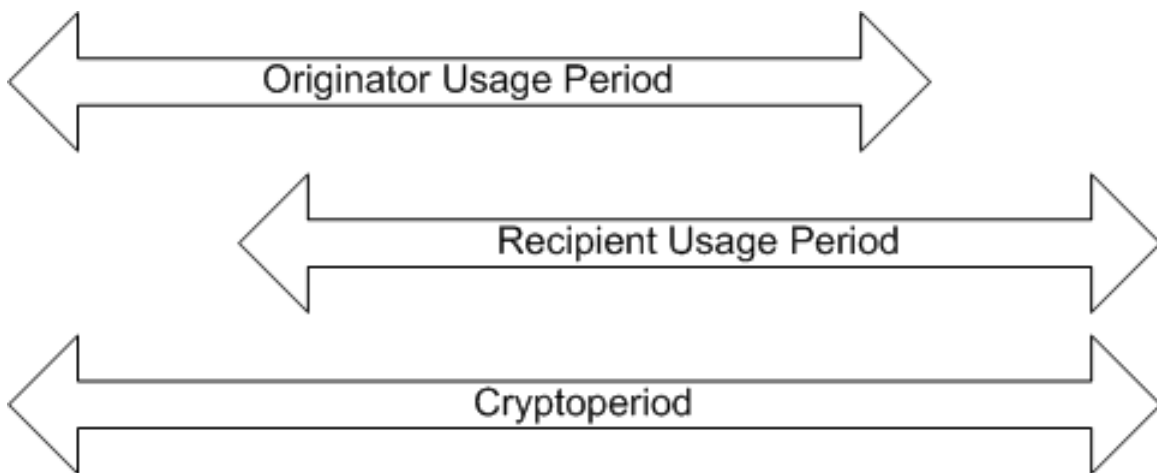
4

5

P1619.3 Key Related Time Periods

6

7



8

9

NIST SP 800-57 Key Related Time Periods

10 The first two time periods, illustrated above, in the P1619.3 standard are a simplification from NIST's
 11 terminology and approach of providing two time periods that may be offset. NIST defines the "originator
 12 usage period" and "recipient usage period".

13 The NIST's originator usage period is the same as P1619.3 standard's Encryption Period. NIST's recipient
 14 usage period might start after the originator usage period. Since storage devices might need to read data
 15 implicitly as soon as it's written, the simplified approach utilized by P1619.3 is used. Therefore, the
 16 simplified P1619.3 diagram is more appropriate for stored data encryption keys.

1 The last two P1619.3 key related time periods (Disable Period and Destruction Period) are not explicitly
2 defined in NIST SP 800-57 because they are directly applicable to encryption of stored data and are not
3 required the ephemeral nature of data protected by encryption of communication links.

4

1 **Annex F**

2 (informative)

3 **Discussion of SO_GUID formats**4 **F.1 Comparison Chart of Name Spaces**

5 Table E.1 shows a quantitative comparison of the various attributes of the allowed SO_GUID formats

6 **E.1 Attributes of Name Spaces**

Attributes	URI	IEEE OUI	Random	EKMI	Zero
Naming Authority	ICANN	IEEE	None	None	None
Reference Documentation	RFC 1034, RFC1035, RFC 3548, RFC 3986,	IEEE Standard 1003.1		EKMI	n/a
Number of SO_Domains in the SO_Family	Greater than 65^{255}	1 *	1 *	1 *	1 *
Number of SO_Contexts in the SO_Family	Greater than 65^{765}	2^{64}	1 *	2^{64}	1 *
Number of SO_Handles in the SO_Family	Greater than 65^{1020}	2^{224}	2^{512}	2^{128}	1*
Fully qualified SO_GUID length in octets **	10 to 1024 ***	28	3 to 66	18	2
SO_Domain length in octets	1 to 255	0	0	0	0
SO_Context ID length in octets	3 to 512	8	0	8	0
SO_Handle length in octets	1 to 255	28	1 to 64	8	0
Vendor Definable Namespace support	SO_Context Path Structure	Yes	No	No	No
Customer Definable Namespace support	SO_Context Path Structure	Yes (potential issues)	No	No	No
Ability to map SO_Handle to SO_GUID within SO_Context	Yes	Yes	Yes	No	No
Multiple SO_Handle definitions supported	Yes	No	No	No	No
Differentiation between SO_Handle and SOGUID for short and long form identification of keys	Yes	Yes	No	Yes	No
Keys sharing between organizations using unique namespace	Yes	Yes	No	No	No

Attributes	URI	IEEE OUI	Random	EKMI	Zero
Who generates the SO_Handle (KM Client, KM Server, KM Client or KM Server, KM Client and KM Server****)	KM Client or KM Server	KM Client or KM Server	KM Client or KM Server	KM Client	KM Client or KM Server

1 * The namespace element is outside the concept for the given SO_Family. In terms of the general object
2 model, this family has only one zero length value for this type of element.

3 ** The SO_GUID length includes the leading 2-octet KM_Family value.

4 *** The smallest fully qualified SO_GUID for the “km” family is “km://-t/x”

5 **** There are potential namespaces that allow both a KM Client and KM Server to establish components
6 of a SO_Handle (e.g. the SO_Handle has a prefix defined by the KM Server and the rest of the SO_Handle
7 by the KM Client)

8 F.2 SO_Family Methods

9 F.2.1 Required methods

10 All SO_Families shall document the following methods.

11 a) form_so_guid:

12 This method shall specify how to form a SO_GUID in the SO_Family given their SO_Family
13 value, a SO_Domain, a SO_Context and a SO_Handle. The SO_Domain, SO_Context, and
14 SO_Handle values and the SO_GUID returned shall be specified using the following data
15 structures:

```

16 typedef unsigned char octet;           // unsigned 8 bit value
17 typedef short int16_t; // signed 16 bit value
18 typedef *octet so_family;           // pointer to 2 octets
19 typedef {
20     octet *value;                   // NULL = no value
21     int16_t length;                 // 0 = no value, < 0 = error code
22 } so_guid;
23 typedef {
24     octet *value;                   // NULL = no value
25     int16_t length;                 // 0 = no value, < 0 = error code
26 } so_domain;
27 typedef {
28     octet *value;                   // NULL = no value
29     int16_t length;                 // 0 = no value, < 0 = error code
30 } so_context;
31 typedef {
32     octet *value;                   // NULL = no value
33     int16_t length;                 // 0 = no value, < 0 = error code
34 } so_handle;
35

```

36 If length is non-negative, then it refers to the length of the value in octets. Note that the value may
37 not be a string that ends in the character ‘\0’ and may contain imbedded ‘\0’ characters. If the value
38 is a ‘\0’ terminated C-style string, then length shall include the trailing ‘\0’ character.

39 The function associated with this method shall be specified as follows:

1 so_guid form_so_guid(so_family, so_guid*, so_context*, so_handle*);

2 b) parse_so_guid:

3 This method shall specify how to parse a SO_GUID in the SO_Family into the SO_Family,
4 SO_Domain, SO_Context, and SO_Handle values:

```
5 typedef {
6     so_family so_family;
7     so_guid *so_guid;
8     so_domain *so_domain;
9     so_handle *so_handle;
10 } so_parts;
```

11
12 The function associated with this method shall be specified as follows:

```
13 so_parts parse_so_guid( so_family, so_guid*);
```

14

15 c) is_valid_so_guid:

16 This method shall specify how to determine if the syntax SO_GUID in the SO_Family is valid and
17 well formed under the rules of the given SO_Family. The function associated with this method
18 shall be specified as follows:

```
19 int is_valid_so_guid( so_family, so_guid* );
```

20

21 The return value shall be the length of the so_guid in octets if it is a valid SO_GUID for the given
22 family, or a value <0 if it is invalid. Negative return value may be used to return a SO_Family
23 specific error code. If the so_family is not valid, 0 shall be returned.

24 d) info_so_guid:

25 This method shall return name/value pairs related to a given a SO_GUID in the context of a given
26 SO_Family. The following name/value pairs shall be returned:

- 27 • When name = "SO_GUID", value = so_guid data structure pointer
- 28 • When name = "SO_GUID.length", value = the SO_GUID length as an int16_t
- 29 • When name = "SO_GUID.value", value = pointer to the SO_GUID octet array
- 30 • When name = "SO_Family", value = so_family data structure pointer
- 31 • When name = "SO_Family.length", value = the SO_Family length as an int16_t
- 32 • When name = "SO_Family.value", value = pointer to the SO_Family octet array of 2 octets
- 33 • When name = "SO_Domain", value = so_domain data structure pointer
- 34 • When name = "SO_Domain.length", value = the SO_Domain length as an int16_t
- 35 • When name = "SO_Domain.value", value = pointer to the SO_Domain octet array or NULL
- 36 • When name = "SO_Context", value = so_context data structure pointer
- 37 • When name = "SO_Context.length", value = the SO_Context length as an int16_t

- 1 • When name = “SO_Context.value”, value = pointer to the SO_Context octet array or NULL
- 2 • When name = “SO_Handle”, value = so_handle data structure pointer
- 3 • When name = “SO_Handle.length”, value = the SO_Handle length as an int16_t
- 4 • When name = “SO_Handle.value”, value = pointer to the SO_Handle octet array
- 5 • When name = “is_valid”, value = a bool where true means it was valid and well formed
- 6 • [TBD]

```

7 typedef {
8     octet *name;           // NULL = no name
9     octet *value;         // NULL = no value
10    int16_t n_length;      // 0 = no name, < 0 = error code
11    int16_t v_length;      // 0 = no value, < 0 = error code
12 } so_data;
13 typedef {
14     so_data *set;         // NULL = no data in set
15     int16_t count;        // 0 = empty data set, < 0 = error
16 } so_info;
17

```

18 The name element shall be a C-Style ‘\0’ character terminated string consisting of only alphanumeric characters, dashes, dots, and underscores. The first octet shall be an alphanumeric character. A name element that begins with a dash, dot, or underscore is reserved for future use by this standard.

21 The n_length element shall include the length of the terminating ‘\0’ character. The minimum length (min n_length value) of the name string shall be 2.

23 The value element is an array of octets that may not be ‘\0’ character terminated and may contain imbedded ‘\0’ characters.

25 The name/value pairs required by this method duplicate the functionality of the other required methods. This additional method is still needed because SO_Families may need to return name/value pair information that is not available in the other required methods.

28 The SO_Family may return additional name/value pairs about the SO_GUID. The SO_Family shall document any additional name/value pairs returned by this method.

30 The function associated with this method shall be specified as follows:

```
31 so_info info_so_guid( so_family, so_guid*);
```

32

33 The SO_Family may document addition methods.

34 This methods and data structures are not an API. They are a specification is given in ANSI C-like pseudo code.

36 **F.2.2 km common methods**

37 **F.2.2.1 km SO_GUID constructor method**

38 *[Content TBD]*

- 1 **F.2.2.2 km SO_GUID parser method**
- 2 *[Content TBD]*
- 3 **F.2.2.3 km valid SO_GUID check method**
- 4 *[Content TBD]*
- 5 **F.2.2.4 km SO_GUID property list method**
- 6 *[Content TBD]*
- 7 **F.2.3 km specific methods**
- 8 *[Content TBD]*
- 9 **F.2.4 na common methods**
- 10 **F.2.4.1 na SO_GUID constructor method**
- 11 *[Content TBD]*
- 12 **F.2.4.2 na SO_GUID parser method**
- 13 *[Content TBD]*
- 14 **F.2.4.3 na valid SO_GUID check method**
- 15 *[Content TBD]*
- 16 **F.2.4.4 na SO_GUID property list method**
- 17 *[Content TBD]*
- 18 **F.2.5 na specific methods**
- 19 *[Content TBD]*
- 20 **F.2.6 rn methods**
- 21 **F.2.6.1 rn SO_GUID constructor method**
- 22 *[Content TBD]*
- 23 **F.2.6.2 rn SO_GUID parser method**
- 24 *[Content TBD]*
- 25 **F.2.6.3 rn SO_GUID valid SO_GUID check method**
- 26 *[Content TBD]*

- 1 **F.2.6.4 rn SO_GUID property list method**
2 *[Content TBD]*
- 3 **F.2.7 rn specific methods**
4 *[Content TBD]*
- 5 **F.2.8 ek common methods**
- 6 **F.2.8.1 ek SO_GUID constructor method**
7 *[Content TBD]*
- 8 **F.2.8.2 ek SO_GUID parser method**
9 *[Content TBD]*
- 10 **F.2.8.3 ek valid SO_GUID check method**
11 *[Content TBD]*
- 12 **F.2.8.4 ek SO_GUID property list method**
13 *[Content TBD]*
- 14 **F.2.9 ek specific methods**
15 *[Content TBD]*
- 16 **F.2.10 00 common methods**
- 17 **F.2.10.1 00 SO_GUID constructor method**
18 *[Content TBD]*
- 19 **F.2.10.2 00 SO_GUID parser method**
20 *[Content TBD]*
- 21 **F.2.10.3 00 valid SO_GUID check method**
22 *[Content TBD]*
- 23 **F.2.10.4 00 SO_GUID property list method**
24 *[Content TBD]*
- 25 **F.2.11 00 specific methods**
26 *[Content TBD]*

1 **F.3 SO_Family Advantages, Potential Concerns and Solutions**

2 **F.3.1 km SO_Family Advantages, Potential Concerns and Solutions**

3 **F.3.1.1 URI Advantages**

- 4 — Ensures globally unique key ids that can be exchanged via the internet in an automated fashion
- 5 using policy and/or access control mechanisms
- 6 — Makes use of existing standards to define namespace format while allowing individual applications
- 7 to decide which Object ID format best suits the applications needs
- 8 — Uses well established, existing naming authority with the ability to limit lookups to local and/or
- 9 remote destinations for keys, policies, etc...
- 10 — Support for legacy key namespaces that do not support a full URI implementation using redirects
- 11 including all of the SO_Family namespaces found in this standard

12 **F.3.1.2 URI Potential Concerns and Solutions**

- 13 — SO_GUID The size is not fixed and can be too large for some implementations
- 14 — It is possible to create a standard translation from stored values within specific applications or on
- 15 media types to a fully qualified SO_GUID or any component of the SO_GUID to allow for
- 16 retrieval of a key via API calls to a KM Client or within a KM Client that a KM Server will
- 17 understand.
- 18 — While a SO_GUID of the 'km' SO_Family can reach a length of 1024 octets an application can
- 19 itself enforce the use of a shorter SO_GUID or through translation

20 **F.3.2 na SO_Family Advantages, Potential Concerns and Solutions**

21 **F.3.2.1 IEEE OUI Advantages**

- 22 — Compact format
- 23 — Many devices use NAA

24 **F.3.2.2 IEEE OUI Potential Concerns**

- 25 — Requires end users to provide their own unique NAA to ensure globally unique identifiers which
- 26 will require additional costs not normally required for end users
- 27 — Use of existing vendor NAA works around this issue without requiring additional expense. Some
- 28 manual configuration of KM Server and key locations would be required to share keys
- 29 between organizations.

30 **F.3.3 rn SO_Family Advantages, Potential Concerns and Solutions**

31 **F.3.3.1 Advantages of Random SO_Handles**

- 32 — The "rn" SO_Family supports existing naming formats for key identifiers including those that were
- 33 not chosen at random. Any existing key identifier that is between 1 and 64 octets in length
- 34 may be used as SO_Handle. Legacy key identifiers may be converted into an "rn"
- 35 SO_Family SO_GUID by preceding it with the two octets: "rn".
- 36 — Random SO_Handles are trivial to generate.
- 37 — Random SO_Handles are anonymous. Their value does not directly reveal information about the
- 38 KM Server under which they were first generated.

- 1 — Because this SO_Family has no naming authority, using random SO_Handles within a single
2 organization is trivial.

3 **F.3.3.2 Potential Concerns of Random SO_Handles**

- 4 — Because this SO_Family has no naming authority, exchanging keys between two independent sets
5 of KM Servers is not trivial. The lack of a SO_Domain and SO_Context in the SO_GUID
6 makes it difficult to locate the KM_Servers that have access to the security object.
7 KM_Servers assume that the KM Server holds the security object, or they must use
8 information that outside of the scope of this standard to locate the security object or they must
9 assume that the security object does not exist.
- 10 — There is no guarantee that a SO_GUID from this SO_Family is globally unique. The chance of two
11 independent entities randomly choosing the same n-octet long SO_Handle (SO_Handle
12 collision) is approximately $256-(n/2)$.
- 13 — Within the same local set of KM_Servers, the KM Server is capable of enforcing SO_handle
14 uniqueness by refusing to store two different keys with the same SO_Handle. One solution to
15 the SO_Handle collision problem is to never share keys outside of the local set of KM
16 Servers. However, one cannot assume that a security object will never need to be shared
17 outside the local set of KM Servers. For example, when two independent sets of KM Servers
18 merge (say because two companies merge) there is a potential for SO_Handle collisions. By
19 mapping “rn” SO_Handles onto different SO_GUIDs of another SO_Family, one may be able
20 to disambiguate the security objects that were previously created in the “rn” SO_Family.
- 21 — The SO_Handle collision is a problem between keys of two independent local sets of KM Servers
22 such as may occur when two independent organizations need to share or exchange keys.
23 There is a chance that a SO_Handle from one organization will collide with key from the
24 other organization. Consider the case where a piece of removal media is encrypted by one
25 organization and then is sent to other organization. If the receiving organization has another
26 key with the same SO_Handle in the “rn” SO_Family, then its KM Clients will almost
27 certainly be unable to decrypt the removable media that they received unless they are willing
28 to replace their existing key with the imported key. One solution to this export problem is to
29 convert exported “rn” SO_GUIDs into a SO_GUIDs of SO_Family that supports a global
30 resolvable namespace. For example, if the “example.com” organization wishes to share the
31 key “rn23209” with an external entity, they can export the key as the SO_GUID
32 “km://example.com/-rn/23209”.
- 33 — Best practices state that for those applications where a proof of SO_GUID uniqueness is not
34 mandatory; a SO_Handle collision of no more than 2-128 is sufficient. Therefore, it is
35 recommended that SO_Handles values contain at least 256 bits of entropy. For example,
36 SO_Handles of at least 32 octets in length that are generated by a cryptographically sound
37 random bit generator would suffice. KM clients that lack the ability to generate such
38 SO_Handles should connect to a capable KM Server and that the KM Server generate the
39 SO_Handle.

40 **F.3.4 ek SO_Family Advantages, Potential Concerns and Solutions**

41 **F.3.4.1 Advantages of EKMI Name Space**

- 42 — Supports existing naming formats for key identifiers

43 **F.3.4.2 Potential Concerns of EKMI Name Space**

- 44 — No guarantee that identifier is globally unique when keys must be shared between organizations
45 — Establishment of naming authority for 64 bit

1 **F.3.5 00 SO_Family Advantages, Potential Concerns and Solutions**

2 **F.3.5.1 Zero Advantages**

3 — This family has only one valid SO_GUID.

4 — The valid SO_GUID of this family is the smallest possible fully qualified SO_GUID.

5 **F.3.5.2 Zero Potential Concerns and Solutions**

6 — The valid SO_GUID does not match any security object. The KM Server shall return an error of a
7 KM Client attempts to resolve the 00 SO_Family SO_GUID.

8