IEEE P1619.3™/D4
Draft Standard for Key Management Infrastructure for Cryptographic Protection of Stored Data

Prepared by the Security in Storage Working Group of the

Computer Society Information Assurance Committee

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

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

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

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

This page is left blank intentionally.

Introduction

[Content TBD]

Notice to users

Laws and regulations

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

Copyrights

Updating of IEEE documents

Users of IEEE standards should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata.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

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

Errata

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

Interpretations

Current interpretations can be accessed at the following URL: http://standards.ieee.org/reading/ieee/interp/index.html.

Patents

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

Participants

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

Matthew V. Ball, *Chair*

Eric Hibbard, *Vice Chair*

Participant1	Participant4	Participant7
Participant2	Participant5	Participant8
Participant3	Participant6	Participant9

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

(to be supplied by IEEE)

CONTENTS

Draft Standard for Key Management Infrastructure for Cryptographic Protection of Stored Data

1. Overview

1.1 Scope

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

1.2 Purpose

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

2. Normative references

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

- NIST Special Publication 800-57 March 2007, Recommendations for Key Management
- ISO/IEC 11770-1:1996 Information technology - Security techniques - Key management - Part 1: Framework
- SOAP v 1.1 (http://www.w3.org/TR/2000/NOTE-SOAP-20000508/)
- IEEE Std 1003.1-2001 Portable Operating System Interface for Computer Environments (POSIX)
- RFC1034 Domain Names – Concepts and Facilities, P. Mockapetris
- RFC3548 The Base16, Base32, and Base64 Data Encodings, S. Josefsson, Ed.
- RFC3986 Uniform Resource Identifier (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter
- RFC4234 Augmented BNF for Syntax Specifications, D. Crocker Editor & P. Overell

3. Definitions, acronyms and abbreviations

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

3.1 Definitions

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

3.1.1 Access control: Restricts access to resources only to privileged entities.

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

3.1.3 Authentication: A process that establishes the origin of information, or determines an entity's identity. In a general information security context: Verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system

3.1.4 Availability: Timely, reliable access to information by authorized entities.

3.1.5 Cipher key: A bit string that controls the pseudo-random permutation of an encryption or decryption routine. See also: cryptographic key.

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

3.1.7 Confidentiality: The property that sensitive information is not disclosed to unauthorized entities. In a general information security context: preserving authorized restrictions on information access and disclosure, including means for preserving personal privacy and proprietary information

3.1.8 Control Plane: In the context encryption of stored data, control plane operations include those operations necessary to identify the encryption requirements of the stored data, generate cryptographic information cryptographic information necessary to properly configure the Cryptographic Units, retrieve this cryptographic information, and configure the Cryptographic Units with the acquired Cryptographic information. In addition, control plane operations include all operations necessary to protect and distribute this cryptographic information throughout a key management environment. Control plane operations tend to be lower performance operations that are decoupled from the data plane where the actual encryption of stored data takes place.

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

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

3.1.10 Cryptographic Unit: An entity capable of using cryptographic algorithms to encrypt data being written to and decrypt data read from storage media. A Cryptographic Unit lies in both the data plane (where encryption and decryption actually take place) and the control plane (where data set information, cryptographic keys, and security parameters are transported between itself and the Key Management Client.

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

3.1.12 Data set: A collection of data, independent of structure, media or transport that are encrypted using a common key.

3.1.13 Data set attribute: A specific attribute such as file path, tape volume id, server IP, a range of disk blocks or database column identifier that can be used to define a policy for how one or more data sets are to be protected.

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

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

3.1.16 Encryption Entity: A software and/or hardware entity that is capable of accepting cryptographic keys and using them to encrypt and decrypt data that is written and read, respectively, to persistent data storage media. Encryption entities are composed of a Cryptographic Unit and a Key Management Client.

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

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

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

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

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

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

3.1.20 Key exchange structure: An xml encapsulation of a key, together with policies governing the use of the key.

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

3.1.22 Key management API: A collection of reference programming interfaces to the Key Management Software Library; one interface each for the C, C++, and Java language bindings.

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

3.1.24 Key management client-server operations: Defines the set of high level operations and their message and transport mechanisms that allow Key Management Clients to request key management services from Key Management Servers and, likewise, to allow Key Management Servers to respond to these key management service requests. These operations utilize a well defined set of key management objects for the communications of key management related information between the clients and servers.

3.1.25 Key management infrastructure: The framework and services that provide for the generation, production, distribution, control, accounting, and destruction of all cryptographic material, including symmetric keys, as well as public keys and public key certificates. It includes all elements (hardware, software, other equipment, and documentation); facilities; personnel; procedures; standards; and information products that form the system that distributes, manages, and supports the delivery of cryptographic products and services to end users.

3.1.26 Key management objects: TBD

3.1.27 Key management server: 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, they may also support security services such has audit services, backup/archive services, security policy management, and access control services.

3.1.28 Key Management Software Library: A software library developed by the P1619.3 Task Group that provides a reference implementation of a portion of the Key Management Client that generates Key Management Client-Server Operations. The interface to the library is the Key Management API. The library uses industry standard SSL/TLS and TCP interface calls for generating Key Management Client-Server Operations over the network. The use of this library is optional, but it can be used to facilitate the rapid development of compliant Key Management Clients by implementers.

3.1.29 Key Management User: One or more entities that are responsible for the configuration and management of the key management environment.

3.1.30 Key pool: A group of keys used for a common purpose. Keys may or may not be used to encrypt the same data. A key may belong to zero or more pools.

3.1.31 Key object: a data structure consisting of the key and its attributes: the key identifier, key contents, generation time, and key to originator binding

3.1.32 Key processing facility: The Key Processing Facility is a KMI component that performs one or more of the following functions:

Acquisition or generation of public key certificates,
Initial generation and distribution of keying material,
Maintenance of a database that maps user entities to an organization's certificate/key structure,
Maintenance and distribution of nodal key compromise lists and/or certificate revocation lists, and
Generation of audit requests and the processing audit responses as necessary for the prevention of undetected compromises.

3.1.33 Key set: A group of keys that were used to encrypt a common set of data at various times. A key may belong to zero or more Key Sets based on data requirements.

3.1.34 KM filename: A filename consisting of only hexadecimal tri-graphs and characters from the POSIX portable filename character set. All characters not part of the POSIX portable filename shall be represented as hexadecimal tri-graphs. No character from a valid POSIX portable filename shall be encoded as a hexadecimal tri-graph.

3.1.35 KM FQDN: A Fully Qualified Domain Name consisting of only hexadecimal tri-graphs and the characters allowed in a domain name as defined by RFC 1034. The RFC 1034 domain name characters are:

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

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

0 1 2 3 4 5 6 7 8 9 .

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

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

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

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

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

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

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

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

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

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

3.1.40 Record ID: A 12 byte octet unique identifier for a security object within a specific SO_Domain. Currently used in the URI name space.

3.1.41 Security object: A collection of data and metadata that controls the cryptographic properties of a different object. Examples of a security object: a cryptographic key, a policy, or set of policies, a security log entry, a user or client description, or authorization.

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

3.1.43 Service agents: Entities that support organizations' KMIs as single points of access for other KMI nodes.

3.1.44 Storage Media: Any device that is used to store and retrieve data over extended periods of time in a persistent manner. [NOTE: We need to check with the IEEE common terms and definitions for a better definition]

3.1.45 Wrap: A process by which data is encrypted and authenticated.

3.2 Acronyms and abbreviations

CN	Client Node
CU	Cryptographic Unit
FQDN	Fully Qualified Domain Name
KPF	Key Processing Facility
KM	Key Management
KM API	Key Management Application Programming Interface
KM Client	Key Management Client
KM Server	Key Management Server
KM SW Lib	Key Management Software Library
KM User	Key Management User
KMCS Ops	Key Management Client-Server Operations
KMI	Key Management Infrastructure
KMS	Key Management Service
KMSS Ops	Key Management Server-Server Operations
PRNG	Pseudorandom Number Generator
RNG	Random Number Generator

SA	Service Agent
SO_GUID	security object globally unique identifier
URI	Universal Resource Identifier

4. General Concepts and Models

4.1 Overview

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

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

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

Therefore this standard specifies a key management infrastructure in which a set of key management servers (KMS) provide key management services to clients. The key management services defined in this standard (see 5) include key distribution, key generation, key availability, key storage, archival and audit.

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

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

The standard also defines a number of key types, including all key types standardized by the IEEE Storage in Security Working Group as of April 2007.

The model of a larger group of clients requesting services from a central set of KMS fits well into the NIST key management model.

4.1.1 Key Management Models

The following subclauses contains a number of abstract models that define the high level architecture for environments and systems that provide management for encryption keys and their related security attributes, parameters, and objects used for the protection of stored data. Industry and international standards define the operations and procedures for managing encryption keys and their related security material as "key management," and thus the objective of these subclauses is to present a series of key management models that provide an introduction to, and a context for, the detailed key management operations, objects, and protocols presented throughout this remainder of this standard.

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

- Key Management Architecture Model – the basic, high level model for key management environments that provides a foundation for the remaining models and detailed information developed in the later sections of this standard.

- Key Management Conceptual Model – provides the basic model for the internal and external organization of the primary components in a key management environment.

- Key Lifecycle Model – the basic state machine model that encompasses the entire set of standard key states and the transitions between these states as maintained by the key management systems that implement this standard.

- Key Management Sequence Models – a series of models that describe the high level interactions between the primary components in the Key Management Conceptual Model.

- Key Management Object Models

- Key Management Operation Models

4.2 Key Management Architecture Model

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

Figure 1 illustrates the architecture model for the key management environment defined in this standard.

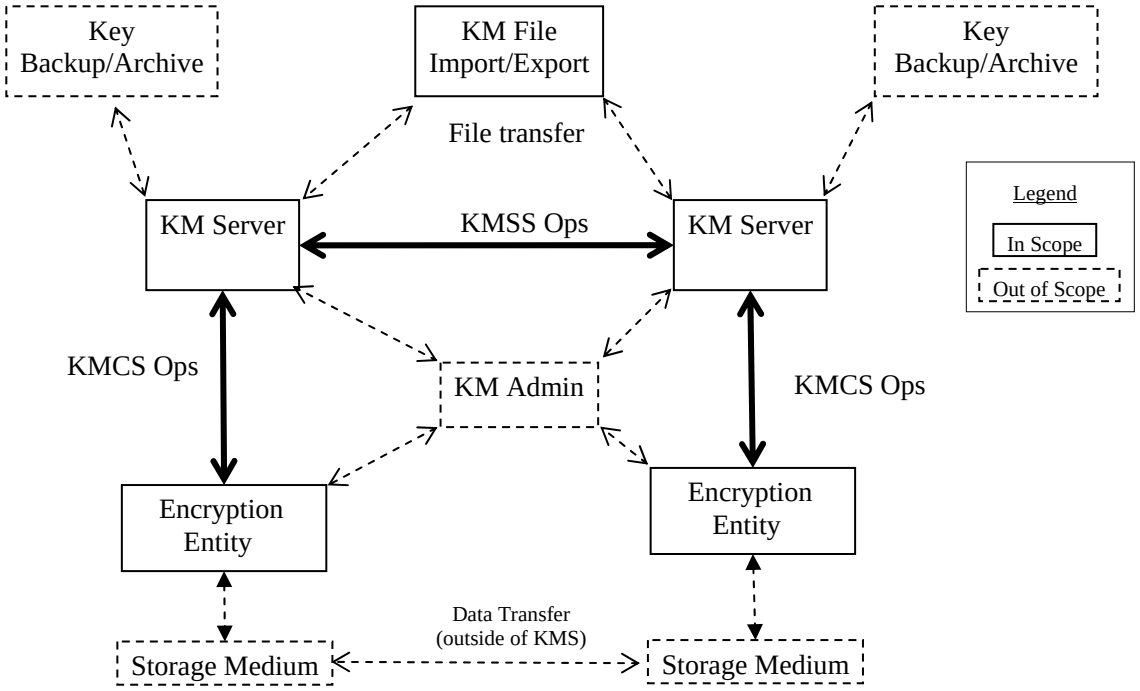


Figure 1 Key Management Architecture Model

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

— KM Server: 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 such has audit services, backup/archive services, security policy management, and access control services.

— Encryption Entity: 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: Any device that is used to store and retrieve data over extended periods of time in a persistent manner. This may include magnetic, optical, or solid-state memory devices (see 4.3).

— Key Backup and Archive: 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: The Key Management File Import and Export service is a hardware and/or software entity that provides file exchange services between KM servers. This service will be defined in a subsequent revision of this specification.

— KM Admin: The key management administrator is an entity that is responsible for the configuration and management of the key management environment. 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: The Key Management Server-Server Operations define the set of high level operations, their messages, and transport mechanisms that allow Key Management Servers to communicate with one another in order to facilitate the delivery of key management services that can not be accomplished with a single Key Management Server. Server-to-server operations facilitate advanced key management services such as hierarchical key distribution and redundancy and availability of key services for key management environments.

— KMCS Ops: The Key Management Client-Server Operations define the set of high level operations, their messages, and transport mechanisms that allow Key Management Clients to request key management services from Key Management Servers and, likewise, to allow Key Management Servers to respond to these key management service requests. These operations utilize a well defined set of key management objects for the communications of key management related information between the clients and servers.

— Scope: The KM Servers, KM Clients, Encryption Entities, KMSS Operations and KMCS Operations are the subject of this document; all other entities are included for completeness, but are nor defined in this documents.

4.3 Key Management Conceptual Model

The key management conceptual model is defined in this standard to provide a more detailed view of the relationship between the critical entities involved in the key management environment. The purpose of the key management conceptual model is fourfold:

- To explicitly identify the differences between data plane operations and control plane operations. The process of actually using an encryption key to encrypt and decrypt data occurs in the data plane. The process of key acquisition and activation occurs in the control plane.

- To identify the entities, components, and their interfaces which are defined as part of this standard.

- To identify the entities and components which are not within the scope of the standard, but need to be understood based on their impact to a key management environment.

- To understand possible embodiments of this standard in actual implementations. The key management conceptual model, coupled with the use cases defined in the informative annex of this standard will aid implementers in their efforts to be compliant to this standard.

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

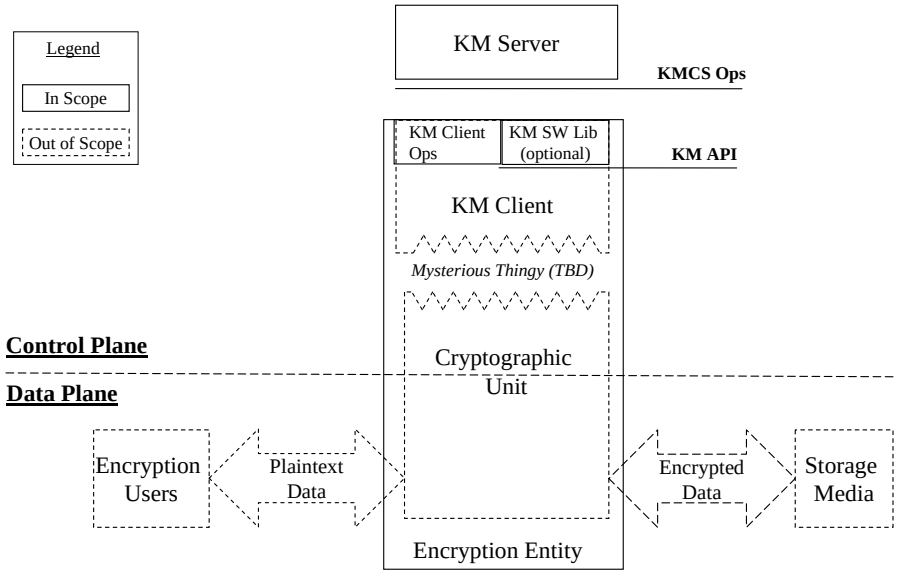


Figure 2 Key Management Conceptual Model

The following list describes each component within Figure 2:

— KM Server: See 4.2

— Control Plane: Where data set information, cryptographic keys, and security parameters are transported between the Cryptographic Unit and the Key Management Client

— Data Plane: Where data encryption and decryption actually take place

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

— Storage Media: Anything capable of storing non-volatile data.

— Encryption Entity: A entity that contains the following logical or physical components:

— KM Client (In Scope): A component of an Encryption Entity that serves several functions:

  i) Communicates with Key Management Servers to acquire cryptographic keys;,

  ii) Communicates with Cryptographic Units to load and activate these keys for their use in encrypting and decrypting data that is written and read, respectively, to data storage media, and;

  iii) Enforces the Key Management Policies as required in this specification.

— The KM Client enforces these functions in one of two ways:

i) KM Client Ops: The key management client enforces these functions through the appropriate implementation and use of the KM Client Operations (out of scope in this document).

ii) KM SW Lib: These functions may be enforced through the use of a standard Key Management Software Library as defined in this document.

— Cryptographic Unit (out of scope): An entity capable of using cryptographic algorithms to encrypt data being written to and decrypt data read from storage media.

4.4 Key Lifecycle Model

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

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

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

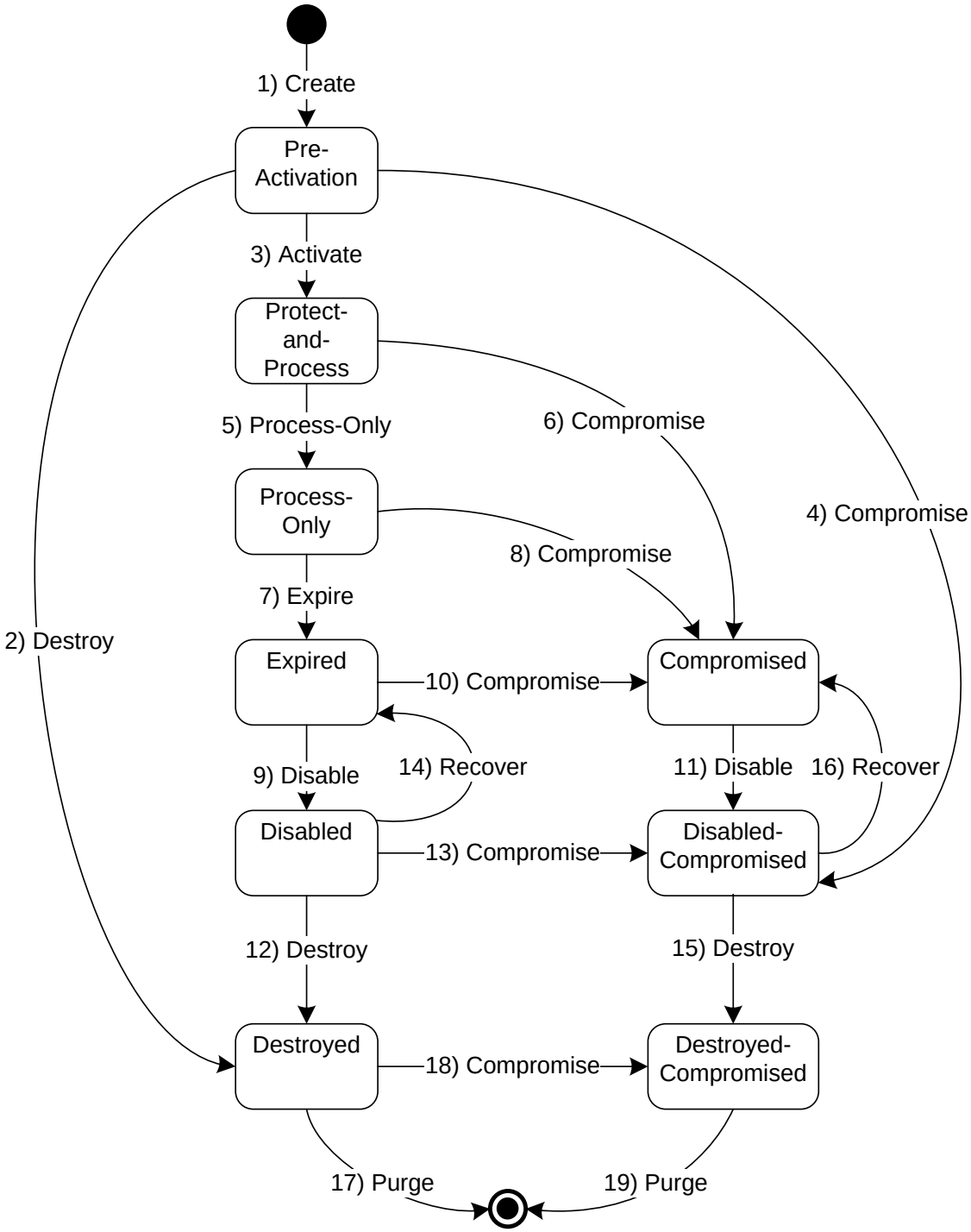


Figure 3 Key Lifecycle Model

The key lifecycle model consists of a set of key states in which depicts the states 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.

4.4.1 Key State Definitions

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

— Pre-Activation: The key has generated but is not yet in use or distributed to any KM Client or Cryptographic Unit. A key in this state can only exist in the KM Server. Such a key can be distributed to a KM Client.

— Protect-and-Process: A key in this state can be used for both encryption (i.e., to protect information) and decryption (to process information). A key is placed into this state when it is initially given to a KM Client, i.e., it is activated. The activation is done when a KM Client requests a new key. If a key is created by a KM Client or Cryptographic Unit and is then given to the KM Server, the key will be immediately placed into Protect-and-Process state. A key will remain in this state until the encryption period passes.

— Process-Only: A key in this state can be used for decryption but not encryption. When a KM Client/Cryptographic Unit determines that none of the keys available to it (e.g., for a specific tape cartridge or disk volume that is being read or written) are in the protect-and-process state, and it needs to perform encryption, it should create a new key. Keys transition from Protect-and-Process to Process-Only when the Encryption Period for the key expires, or as a result of an administrative action. A key will remain in the Process-Only state until the Crypto Period passes. At that time the key will be expired and move to the Expired state.

— Expired: An expired key is a key that has had its Crypto Period expire, but it may still be needed to process (decrypt) information. Keys in this state may be used to process (decrypt) data only. An expired key can be delivered to a KM Client, but the Cryptographic Unit should not use the key to encrypt data. A key will remain in this state until the key's Disable Period expires or as a result of an administrative action. At that time the key will be disabled and it will move to the Disabled state.

The difference between keys in the Process-Only and Expired states is subtle. As far a KM Clients or Cryptographic Units are concerned, the states are equivalent. The difference is significant only to KM Servers. A key in the Process-Only key is still in the NIST operational phase and would routinely be delivered to KM Clients. A key which has had its Crypto Period expire has moved into the NIST post-operational phase, and may have additional restrictions imposed on its delivery to KM Clients.

— Disabled: A disabled key is a key that is still known to the KM Server but it will not be delivered to a KM Client. A disabled key's key material remains intact in the KM Server. A key will remain in the Disabled state until either the key's Destruction Period expires, or administrative action places the key in the Destroyed state, Compromised state, or recovers the key to the Expired state. If the key's Destruction Period expires, the key will be destroyed and will transition to the Destroyed state.

— Compromised: Keys are compromised when they are released to or discovered by an unauthorized entity. Compromised keys should not be used to protect information, but may need to be used to process information. The KM Server cannot determine if a key has been compromised. An administrative action is needed to inform a KM server that a key has been compromised. A compromised key will be delivered to a KM Client, but the Cryptographic Unit should not use the key to encrypt data. Just as with Expired keys, a KM Server may impose additional restrictions on the delivery of compromised keys to a KM Client. A key will remain in this state until the Disable Period expires. At that time the key will be disabled and move to the Disabled-Compromised state.

— Disabled-Compromised: A key that is both disabled and compromised. Disabled and compromised keys will never be delivered to KM Clients. A key will remain in the disabled compromised state until either the Destruction Period expires or the key is recovered to the Compromised state by administrative action. If the Destroy Period expires, the key will be destroyed and moved to the Destroyed-Compromised state.

— Destroyed: A destroyed key is a key which has had its key material removed from the KM Server. Information or metadata about the key may be retained by the KM Server. Destroyed keys cannot

be delivered to a KM Client. Keys can be destroyed by either an administrative action in the KM Server, or by the expiration of the Destruction Period of the key while it is in the Disabled or Disabled-Compromised states.

— Destroyed-Compromised: Similar to keys in the Destroyed state, but the key was compromised before or after destruction.

— Terminal (Purged): Purged is the terminal state for keys. A purged key is a key that no longer exists in the KM Server in any form. Neither the key material nor any metadata about the key is known to the KM Server. A purged key cannot be delivered to a KM Client.

4.4.2 Key State Transition Definitions

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

— (1) Create: When a key is created by a KM Server, it begins in the pre-activation state. This transition occurs as soon as a key is generated within the KM Server, such as a key being generated by a random number generator (RNG).

Transition 1 applies to newly created keys; the key immediately transitions to Pre-Activation state upon its creation.

— (2) Destroy: A key in the Pre-Activation state may be moved directly to the Destroyed state. If the key has never been activated and is no longer required, but there is a requirement to maintain information about the key, the key may be destroyed. This transition can only occur as a result of administrative action.

Transition 2 applies to keys in the Pre-Activation state; the key transitions to the Destroyed state.

— (3) Activate: A key transitions from the Pre-Activation state to the Protect-and-Process state when it is available for use. This transition occurs when the key is assigned for use by a KM Client. There is a bit of confusion around the terminology, since a KM Client may view this action as "creating a key" even though the KM Server views this as assigning a previously generated key.

Transition 3 applies to keys in the Pre-Activation state; the key transitions to the Protect-and-Process state.

— (4) Compromise: A key transitions to the Compromised state when it has been determined to have been released to or discovered by an unauthorized entity. The KM Server cannot determine that this has happened, so this transition occurs as the result of an administrative action.

Transition 4 applies to keys in the Pre-Activation state; the key transitions to the Compromised state.

— (5) Process-Only: A key transitions from the Protect-and-Process state to the Process-Only state when a period of time equal the Encryption Period has expired after the key is activated. This transition may also occur as a result of administrative action. Some KM Clients and Cryptographic Units may be unable to strictly enforce this transition.

Transition 5 applies to keys in the Protect-and-Process state; the key transitions to the Process-Only state.

— (6) Compromise: Same actions as for the (4) Compromise transition.

Transition 6 applies to keys in the Protect-and-Process state; the key transitions to the Compromised state.

— (7) Expire: A key transitions from active to Expired when its Crypto Period expires. This transition may also occur as a result of administrative action.

Transition 7 applies to keys in the Process-Only state; the key transitions to the Expired state.

— (8) Compromise: Same actions as for the (4) Compromise transition.

Transition 8 applies to keys in the Process-Only state; the key transitions to the Compromised state.

— (9) Disable: This transition will occur after the disable period for a key passes. This transition can also occur as a result of administrative actions. The key material and its metadata will be retained by the KM Server. The key will no longer delivered to KM Clients.

Transition 9 applies to keys in the Expired state; the key will transition to the Disabled state.

— (10) Compromise: Same actions as for the (4) Compromise transition.

Transition 10 applies to keys in the Expired state; the key transitions to the Compromised state.

— (11) Disable: Same actions as for the (9) Disable transition.

Transition 11 applies to keys in the Compromised state; the key transitions to the Disabled-Compromised state.

— (12) Destroy: A key can be destroyed when it is no longer needed. When the Destruction Period for a key expires, it will be destroyed. This transition can also occur as a result of administrative action. Active keys (keys in the Protect-and-Process or Protect-Only states) cannot be destroyed; only keys that have been previously activated and are now in the Disabled or Disabled-Compromised states can be destroyed. When a key is destroyed, the key material is removed from the KM Server. Metadata about the key is retained in the KM Server.

Transition 12 applies to keys in the Disabled state; the key transitions to Destroyed state.

— (13) Compromise: Same actions as for the (4) Compromise transition.

Transition 13 applies to keys in the Disabled state; the key transitions to Disabled-Compromised state.

— (14) Recover: When a disabled key is required to be delivered to KM Clients, it can be recovered. This occurs as a result of an administrative action.

Transition 14 applies to key in the Disabled state; the key transitions to Expired state.

— (15) Destroy: Same actions as for the (12) Destroy transition.

Transition 15 applies to key in the Disabled-Compromised state; the key transitions to the Destroyed-Compromised state.

— (16) Recover: Same actions as for the (14) Recover transition.

Transition 16 applies to keys in the Disabled-Compromised state; the key transitions to the Compromised state.

— (17) Purge: The action of purging a key removes all information about the key from the KM Server's key records. Neither the key material nor the key value will be retained by the KM Server. Note, however, that audit logs or other indirect information in the KM Server may still contain information about the key.

Transition 17 applies to keys in the Destroyed state; the key transitions to the Purged state. However, since the key does not exist in the KM Server, there is no record for the key showing the Purged state.

— (18) Compromise: Same actions as for the (4) Compromise transition.

Transition 18 applies to keys in the Destroyed state; the key transitions to the Destroyed-Compromised state.

— (19) Purge: Same actions as for the (17) Purge transition.

Transition 19 applies to keys in the Destroyed-Compromised state; the key transitions to the Purged state. However, since the key does not exist in the KM Server, there is no record for the key showing the purged state.

4.4.3 Key Time Period Definitions

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

The key life cycle is based on four time periods:

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

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

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

[mention that these are not the only time periods available – these are the time periods that can cause a key state transition.]

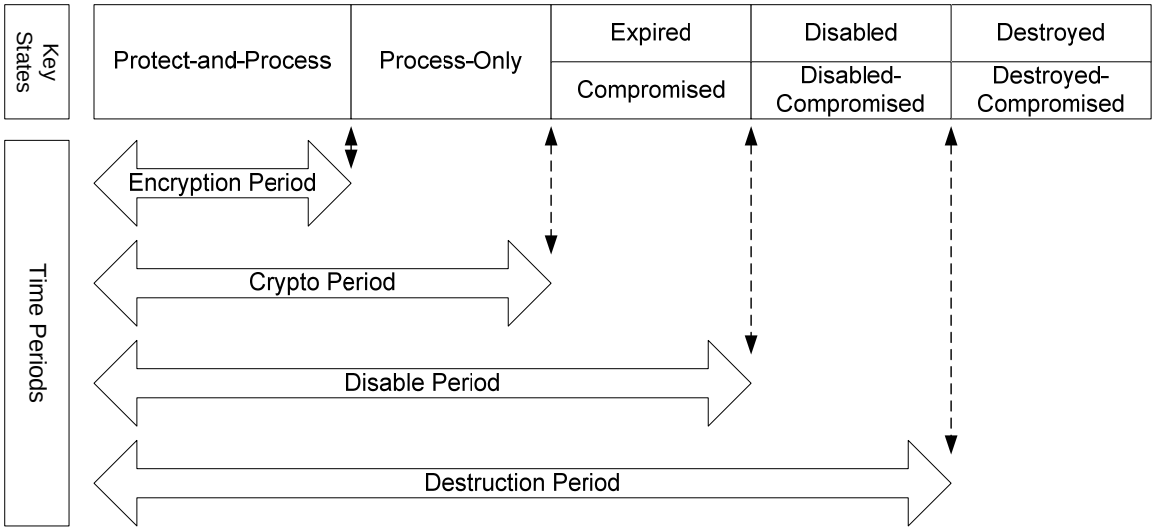


Figure 4 Key Time Periods Compared to Key States

4.5 Key Management Sequence Models

[Content TBD]

4.6 Key Management Operations Model

Another important aspect of this key management standard is the conformant operations that are defined to take place between a Key Management Client (KM Client) and a Key Management Server (KM Server). A

key management operation model is presented here to illustrate the high level flow of management information between these KM Client and KM Server entities. The general sequence of operations, notions of synchronous versus asynchronous operations, and support for single threaded versus multi-threaded operations are critical the interoperability of conformant implementations of this standard.

[More text and diagrams in development.]

4.7 Key Management Object Model

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

[Need D1 Word source to copy object model definitions here.]

5. Key Management Namespace

5.1 Globally unique identifier for Key Management Services

5.1.1 Overview

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

A SO_GUID shall contain the following information, in order:

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

The SO_Family shall be a value from Table A:

Table 1 SO_Family values

SO_Family	Description	Name Authority	Ref.
‘km’	A format based on a URI name space	ICANN	4.4.3
‘na’	A format based on an IEEE OUI name space	IEEE OUI	4.4.4
'rn'	A format based on 32 octet random numbers	None	4.4.5
‘ek’	A format based on the EKMI name space	None	4.4.6
‘00’	A SO_GUID that does not match any object	None	4.4.7

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

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

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

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

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

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

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

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

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

The SO_GUID shall consist of the following components:

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
```

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

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

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

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

The Family_Space is used to map a SO_GUID from a non-“km” SO_Family to the “km” SO_Family. The two octets following the dash shall be a valid SO_Family other than “km” or “00”. The Family_Space “-km” is not a valid Object_Space in the “km” SO_Family. The Family_Space “-00” is not a valid Object_Space in the “km” SO_Family. When a valid SO_GUID in the km SO_Family has an Object_Space that is a Family_Space, the SO_Handle shall be the SO_GUID of the referenced SO_Family, is a KMS symlink whose value is the SO_Handle.

An SO_GUID in the km SO_Family with an “r” Object_Space shall always refer to a KMS symlink whose value refers to a SO_GUID in the same SO_Domain.

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

The Reserved_Space is reserved for future use by this standard.

The SO_Context is unique within its SO_Domain.

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

The SO_Handle is a unique value within a given SO_Context that is under a given SO_Domain. The SO_Handle is a KM filename. A SO_Handle identifies a security object (i.e., a key, policy, key set, key pool, client, log, record number, etc.)

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

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

Table 3 Required Object_Space values for various Security Objects

Security Object Type	Object_Space value
Key	“k” “e” “y”
KMS Policy	“p” “o” “l” “i” “c” “y”
Key Management Realm	“r” “e” “a” “l” “m”
Key set	“s” “e” “t”
Key Pool	“p” “o” “o” “l”
KM Client	“c” “l” “i” “e” “n” “t”
KMS Log	“l” “o” “g”
Record ID	“r”
PKCS#11	“p” “1” “1”

Security objects with may be referenced in several ways:

2) Globally by: SO_Family, SO_Domain, SO_Context and SO_Handle

3) SO_Context and SO_Handle within the clients current SO_Domain
4) SO_Handle within a clients current SO_Context

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

1) km://kms.example.org/key/dir1/dir2/key123
2) km://example.com/key/dir1/%00%00%EA%05
3) km://kms.bigbank.example.com/key/000102030405060708090A0B0C0D0E0F
4) km://example.net/policy/bizpolicy/storsecpolicy/kmspolicy/keypolicy3
5) km://kms.subdomain.example.com/client/location/kmclient1
6) km://kms.division.bigcompany.example.com/realm/Hayward
7) km://dc4.bigfirm.example.biz/.product.example.com/legacy_key_from_old_product
8) km://dc4.bigfirm.example.biz/.acmeprod.example.biz/200708/acme_product_leyacy_key
9) km://prod.example.info/-rn/legacy/server1/%02%32%09

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

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

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

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

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

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

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

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

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		

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

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

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

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

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

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

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

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

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

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

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

<SO_GUID> = <SO_Family> “://” <SO_Handle> /

a) <SO_Family> = “ek”

- “ek” shall be followed by “://” when preceding a <SO_Handle>

b) <SO_Handle> = <GKID>

- <GKID> = <DID> 0x2D <SID> 0x2D <KID>
- <DID> = DIGIT 1*8
- <SID> = DIGIT 1*8
- <KID> = DIGIT 1*8

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

- A SO_Handle identifies a security object - i.e., key

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

- 0-0-0
- 1-2-3
- 10514-22-344342232
- 18446744073709551615-18446744073709551615-18446744073709551615

[Content TBD]

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

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

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

6. Key Management Objects

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

6.1 Key

Scope: Client & Server.

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

6.1.1 Attributes

A key object distributed by a KMS contains the following attributes:

— **KEY_ID** (Type: SO_GUID)

— **FRIENDLY_NAME (**Optional: Type:String) Not necessarily unique within a KMS as additional attributes may be used to make a unique reference.

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

  —

— **STATE** (Type:String) EDITORIAL: Reference back to the relevant section.

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

— T_DISABLED (Type: UTC - time beyond which the key should be used)

- T_CACHED (Type: 64-bits – seconds that the key may be cached for. This may be differentiated by endpoint)
- **CIPHER_TYPE** (Type:String OID – **TODO:** Insert)
- KEY_BLOB (Object as defined in the next section) (This may be differentiated by endpoint, and is constructed from an immutable key value, the storage and representationof which is outside of this standard)
- *VENDOR_SPECIFIC_EXTENSIONS*
- *APPLICATION_EXTENSIONS*
- *CACHING_POLICY*
- (VERSIONING_INFORMATION:)
  - Narrative: A KMS shall represent versioning of Key objects using two values: Version, which is an incrementally increasing value, representing changes to Key attributes, including changes to policies referenced by the key, and a GMT dateTime value representing the time of the last change. KMS_Clients may treat the combination as an opaque token, or use the values to protect against updates of stale copies. KMS servers may construct these values customized to the requestor, or maintain them globally independent of the endpoints. (for example, if a policy for a key changes, but that change is not relevant for an endpoint, the KMS may or may not represent update the versioning_Information. Versioning information will not change due to auditing activity, reference, inclusive Realm_Associations or backup.
  - VERSION (Type: unsigned Numeric)
  - (NOTE: It must be possible for an endpoint to request key object if altered since a reference version)
  - EDIT_DATETIME (TYPE: DATETIME)
  - (NOTE: It must be possible for an endpoint to request a list of key objects altered since a reference time)
-

In addition, a KMS must be capable of representing the following attributes and references:

- REALM_ASSOCIATIONS
  - (Note: keys will not be delivered to endpoints without compatable Realm rights)
- *WRAPPING_POLICY*
- *DESCRIPTION* (Type:String)
- SOURCE
  - *Represents the identity of the originator of the key value. (a specificKMS, a specific client, a specific PEP)*
- *ATTRIBUTE_ASSOCIATIONS*
  - (A list of named value pairs useable as an alternate mechanism to define or reference a key . Keys can be retrieved by their key_ID or alternatively by an ANDED match on these NVPs)

In addition, a KMS must be capable of representing the following associations:

- *USE_BY_CLIENTS*
  - *(Note: does not alterVersioning_Information for a key itself. Note since any given key may be used by multiple clients or PEPs, this tracking must be maintained, so the KMS is capable of initiating unsolicited updates to a KMS_Client when a key's attributes or policies change)*
- *USE_BY_PEPS (see above note)*

6.1.2 States

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

6.1.3 Operations

- Create
- Get
- Store

6.2 Key Blob

6.2.1 Attributes

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

Editorial: CMS will be used to wrap keys. The updates necessary to add key wrapping will be done at a later point.

6.3 Key_Template

Scope: Client & Server.

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

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

Discussion: I suppose one could think of “template” as an additional “dataset binding” and use the same service as previously envisioned. The important notion here is the ability to predefine all the policy and attributes into some template to avoid having to manage all this separately.

6.3.1 Attributes

A Key_Template object defined within a KMS contains the following attributes:

- **KEY_TEMPLATE_ID** (Type: SO_GUID)
- **FRIENDLY_NAME (**Optional: Type:String) Unique within a KMS
  - Note: It should be possible to request a key creation by template using its Friendly_name
- **CIPHER_TYPE** (Type:String OID – **TODO:** Insert)
- *VENDOR_SPECIFIC_EXTENSIONS*
- *APPLICATION_EXTENSIONS*
- *CACHING_POLICY*

- (VERSIONING_INFORMATION:)
  - VERSION (Type: unsigned Numeric)
  - EDIT_DATETIME (TYPE: DATETIME)
- REALM_ASSOCIATIONS
- *WRAPPING_POLICY*
- *DESCRIPTION* (Type:String)

6.4 ENDPOINT_TYPE

Endpoint_Type is an object to simplify the need to exchange capabilities between a KMS_CLIENT or PEP and a KMS_SERVER, as well as managing a collection of capabilities at the Server. (Discussion point: It would be desirable if these values were standardized in some registry.) An Endpoint_Type will always equate to a deterministic set of capabilities, though the converse need not be true. During registration, KMS_Clients or PEPs will present identifying information that will allow a KMS_Server to map it to an Endpoint_Type.

6.4.1 Attributes

- ENDPOINT_TYPE_ID (Type:TBD)
- CAPABILITIES (Note: common registry should allow retrieval of such characteristics as has-certificate, understands-time, min and max keyID lengths, never-exposes-key, hasHSM, etc.)

6.5 REALM (optional)

Realms are used to segment objects into separate administrative domains.

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

When a KMS provides Realm support, the KMS must insure no object is assessable unless the requesting endpoint or administrator has been properly associated with an incorporating realm that allows the access. Realms must allow the following distinctions: create, edit, delete, read, reference (use but not view). The most privileged right from any associated realm may be use to determine an access.

6.5.1 Attributes

- REALM_ID (Type:Realm://domain/realm-name where domain is a string that is in compliance with DNS name as defined by RFC 1034.
- DESCRIPTION

6.6 PEP (Policy Enforcement Point / Cryptographic Unit)

Scope: Client & Server

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

6.6.1 Attributes

- PEP_ID (Type:SO_GUID)
- Array of Name, Value pairs.

NOTE: The name of the attributes that are part of the PEP object shall follow the following convention:

pep://domain/context/attribute-name where domain is a string that is in compliance with DNS name as defined by RFC 1034.

In addition – the following domin/context combination – ieee.org/siswg/ is reserved for use by this standard.

- REALM_ASSOCIATIONS (Server Only)
- ENDPOINT_TYPE_ID (TYPE:TBD Server Only.)
  - DiscussionPoint: Prefer this to be some IEEE registry. note: KMS Clients shall provide a CIM_PhysicalElement or CIM_SoftwareIdentity object upon registration of a PEP, which will allow a KMS to map an entity to a TYPE_ID. Or perhaps we define a new CIM object derived from common ones to include capabilities we want that may not be determined by attributes in the mentioned CIM objects.
- CLIENT_ASSOCIATIONS (Server Only)
  - Note this might be done with a Client_Group. A PEP will initially be associated with full rights granted to its registering Client. Other behaviors to manage associations is outside the scope of this spec., but a KMS must be capable of conforming to a policy or configuration that prevents a PEP from receiving its key from an "unauthorized" client. This can easily be accomplished by restricting access to keys both to authorized KMS clients and authorized PEPs.)
- *AUTHENTICATION_POLICY*
- *AUTHENTICATION_VALUES (It must be possible for a PEP to authenticate to the KMS through an untrusted KMS_CLIENT)*
- *List of {CREDENTIAL_LENGTH,CREDENTIAL_VALUE} tuples.*
- *WRAPPING_POLICY. (Note: this may typically be inherited via endpoint_type_id)*
- *WRAPPING_VALUES (Since PEPs can uniquely protect some wrapping values, such as private keys, data can pass through a KMS_Client without exposure)*

6.7 Client

Scope: Client & Server.

The client object consists of its credentials and capabilities.

6.7.1 Attributes

- CREDENTIAL_TYPE (Session, Username/Password, Symmetric / Asymmetric key, SSO, CHAP)
- List of {CREDENTIAL_LENGTH,CREDENTIAL_VALUE} tuples.
- REALM_ASSOCIATIONS
- **ENDPOINT_TYPE_ID**
- WRAPPING_POLICY

—

6.7.2 States

— Active
— Disabled/Locked
— Authenticated

6.7.3 Operations

— Create
— Delete
— Authenticate
— Disable

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

6.8 Capability

6.8.1 Attributes

— Name (Type: String)

6.8.2 States

None.

6.8.3 Operations

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

6.9 Data Sets

Scope: Client, PEP & Server.

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

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

6.9.1 Attributes

— NAME
— VALUE
— SIZEOF_VALUE

6.10 Client Groups

Scope: Server only.

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

6.10.1 Attributes

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

6.11 Key Groups

Scope: Server only.

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

6.11.1 Attributes

- TYPE (STATIC or DYNAMIC)
- List of CLIENT_SO_GUIDs (only in case of static binding)
- List of {PATTERN, DATASET} tuple.
- REALM_ASSOCIATIONS

7. Key Management Policies

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

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

7.1 Key Assignment Policy

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

A key assignment policy determines

- the type of unencrypted data that is determined to be encrypted with a specific key.
- how often to generate new keys

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

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

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

container attribute name = ”storage_server_name” value=”Ireland.com”

data attribute name = ”financial”

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

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

7.1.1 Attributes

- KEY_ASSIGNMENT_POLICY_ID
- DESCRIPTION
- REALM_ASSOCIATIONS

7.2 Retention Policy

7.2.1 Overview

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

This policy should be superseded by the key life cycle.

7.2.2 Attributes

- RETENTION _POLICY_ID
- DESCRIPTION
- REALM_ASSOCIATIONS
- T_EXPIRATION
- T_DISABLE

7.2.3 States

- Created
- Assigned
- Enforcing

- Disabled

7.2.4 Operations

- Add
- Associate
- Disassociate
- Delete

7.3 Wrapping Policy

The wrapping policy indicates whether a key should be wrapped prior to being dispatched to a client. This policy may be referenced from the key object, ?a key_group?, a client object, ?a ClientGroup?, a PEP object, or a Endpoint_type. If multiple policies are defined then the order of precedence shall be per the following:

1. Key shall prevail over KeyGroup
2. Key or KeyGroup shall prevail over PEP
3. PEP shall prevail over Pep's Endpoint_Type
4. Client shall prevail over Client's Endpoint_Type
5. If both a PEP and Client specify (or inherit) a Wrapping Policy, the key will be double wrapped, first by policy prevailing for the PEP, then by prevailing ClientPolicy. The KMS_Client will unwrap the key and pass the wrapped Client_Key for subsequent unwrapping.

7.3.1 Attributes

- WRAPPING_TYPE (asymmetric / symmetric / signature)
- WRAPPING_MODE (as listed in the key blob section)

7.3.2 States

- Created
- Assigned
- Enforcing
- Disabled

7.3.3 Operations

- Add
- Associate
- Disassociate
- Delete

7.4 Audit Policy

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

[Content TBD]

7.4.1 Attributes

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

NOTE: The only mandatory type that should be supported is ‘log’ and the mandatory dispatch destinations are local and syslog.

7.4.2 States

- Created
- Assigned
- Enforcing
- Disabled

7.4.3 Operations

- Add
- Associate
- Disassociate
- Delete

7.5 Access/Distribution Policy

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

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

This policy is referenced by a key or key group.

Note: this policy governs what endpoints MAY receive a key, but can not be used to determine which endpoints in fact received any given key.

7.5.1 Attributes

- SO_GUID
- Client or clientGroup list
- PEP list
- KM server list.
- REALM_ASSOCIATIONS

7.5.2 States

- Created
- Assigned
- Enforcing
- Disabled

7.5.3 Operations

- Add
- Associate
- Disassociate
- Delete

7.6 Caching Policy

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

Attributes

- CACHING_TYPE, T_CACHE_INTERVAL tuples (list)
- Note: It should be possible to allow different time intervals for caching depending on the security of the caching along different attributes such as HSM, TPM, neverExposedHardware,

7.6.1 States

- Created
- Assigned
- Enforcing
- Disabled

7.6.2 Operations

- Add
- Associate
- Disassociate
- Delete

8. Key Management Operations

8.1 Register Endpoint

Scope: KMCS Ops, including registration for KMS_Client or PEP

[Content TBD]

We need to fill this operation out. e.g. Identify who is registering, type and/or capabilities, how authenticate, certs, etc. send cim object

8.2 Authenticate

Scope: KMCS

8.2.1 Overview

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

8.2.2 Input / Output / Error

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

8.3 Capability Negotiation

Scope: KMCS

8.3.1 Overview

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

8.3.2 Input / Output / Error

- (I): Client
- (I): List of Capability Objects
- (O): List of Capability Objects that are supported by the KM server

8.4 Get Server Capabilities

- (I): Client
- (O): List of Capability Objects.

8.5 Create/Generate Key

Scope: KMCS, KM Console

8.5.1 Overview

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

8.5.2 Input / Output / Error

- (I): Client
- (I) *PEP_ID* or EndPoint's CIM Object - Identifier of final destination for the key.
- (I): List of Dataset objects.
- (O): Key (including unique So_Guid)
- (E): …

8.6 Store Key

Scope: KMCS, KM Console

8.6.1 Overview

Keys that are generated at the client can be stored in the KM server by invoking its store functionality.

8.6.2 Input / Output / Error

(I): Client

(I): List of Dataset objects.

(I) *PEP_ID* or EndPoint's CIM Object - Identifier of final destination for the key.

(O): Key_SO_GUID

(I) *Friendly_Name*

(E)…

8.7 Get Key

8.7.1 Overview

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

8.7.2 Input / Output / Error

(I): Client

(I) PEP

(I): List of Dataset objects.

(O): Key

(E): …

8.8 Push Audit Message

8.8.1 Overview

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

8.8.2 Input / Output / Error

(I): Client or PEP

(I): Key_SO_GUID

(I): Message (Optional)

(O): Boolean – SUCCESS/FAILURE.

(E): …

8.9 Get Random Bytes

8.9.1 Input / Output / Error

- (I): Client (question: why?)
- (I): numbers of bytes desired
- (O) Base64 Encoded bytes

8.10 GetStatus --KMS_Client Service (optional) -- [Server initiated]

Server asking client for status.

Discussion: To aid KMS Administrators to provide key management for endpoints, we will need some mechanism to gather status on endpoints, including operating mode, Keys in use, status of any rekeying, We could try and define some predetermined status types or just allow these to come back with what the KMS Client and PEP can provide.

Discussion: This probably matches the concepts of some existing WSMAN service.

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

8.11 UpdatePending --KMS_Client Service (optional) [Server initiated]

Server notifying client of relevant updates.

KMS_Clients expose this service. KMS Servers will repeat this service until the client acknowledges currency by virtue of matching UpdateVersioningTokens. Note, these tokens are used for sequencing between this service and GetChangeList. A simple implementation of this would be for the KMS Server to maintain a VersioningToken to represent the latest version of "Everything" for a KMS_Client or PEP and a response to a GetUpdateList that returns everything.

- (I): Type (Keys, AllKeys and custom types)
- (I) Scope: (KMS_Client or PEP identifier)
- (I) UpdateVersioningTokens - KMS server sends its tokens representing the state to which the client (or PEP) needs to update.
- (O) UpdateVersioningTokens - KMS client sends its tokens representing the state it has received
-

8.12 GetUpdateList --KMS_Server Service [Client initiated]

Discussion: can this be done with a WS-ENUMERATE service?

- (I): Type: (Keys, AllKeys and custom types)
- (I) Scope: (KMS_Client or PEP identifier)
- (I) UpdateVersioning Tokens (zero values will result in all requested instances of requested type)
- (O) requested objects
- (O) New UpdateVersioningTokens

9. Key Management Transport

9.1 SOAP based protocol

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

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

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

An xml schema definition and WSDL [will be provided].

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>
```

9.1.1 getMethods request message

Purpose: Retrieves a list of valid RPC method names.

Parameters: None.

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>
```

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>
```

9.1.2 getKeyByStorageAttributes request message

Required Parameters:

— DataSetBindings

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

Optional Parameters:

— Key Classname and Key Format

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

ReplicationPolicy

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

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>
```

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>
```

9.1.3 getKeyByID request message

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

Required parameter:

— Key ID portion of the Key element

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>
```

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>
                    <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>
```

9.1.4 QueryKey request message

Purpose: Find keys by attributes known to the client (e.g. data set).

Required parameters:

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

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

Example: A query for a key of a particular type, requested by a particular user, and used to encrypt one or 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>
```

Response: The KMS responds with two key exchange structures that match the criteria. Only one of 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>
```

9.1.5 storeKey request message

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

Required parameters:

—Key Object parameter

Optional parameters:

—All of the key policy elements may be sent

Result and response: The key will be inserted into the KMS with the supplied policies. If the key already exists in the KMS, the command will fail.

NOTE— Parameter checking will be performed by the KMS and appropriate error returned for 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>
```

Return parameters: The success message is returned on successful key store. Note, this method will 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>
```

9.1.6 updateKeyInfo request message

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

Required parameters:

— Key ID component of the Key Object parameter

Optional parameters:

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

Result and response:

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

NOTE—Parameter checking will be performed by the KMS and appropriate error returned for malformed requests. Caution should be taken when updating data set bindings.

9.1.7 generateKey request message

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

Required parameters:

— Key Classname and Key Format fields.

Optional parameters:

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

Result and response:

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

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>
```

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>
```

9.1.8 getSeed request message

Purpose: The client requests 64 bytes of entropy.

Required Parameters:

— none

Optional Parameters:

— none

Result and response:

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

Prerequisites:

Client should have been authenticated with KMS prior to the request.

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>
```

Return params: KMS 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>
```

9.2 Binary command based protocol

This section describes a lightweight binary protocol between the client and KMS. This protocol will support the above mentioned commands.

9.2.1 Overview

[Content TBD]

9.2.2 Get key by storage attribute

[Content TBD]

9.2.3 Get key by ID

[Content TBD]

9.2.4 Query key

[Content TBD]

9.2.5 Store key

[Content TBD]

9.2.6 UpdateKey info

[Content TBD]

9.2.7 Generate key

[Content TBD]

9.2.8 GetSeed

[Content TBD]

10. Key Management Messaging

[Content TBD]

Annex A

(informative)

Bibliography

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

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

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

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

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

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

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

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

Annex B

(informative)

Example Use Cases

B.1 Tape libraries with encrypting drives

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

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

— KMS one time setup operations

1) Client Registration – the KMS administrator enrolls the library as a KM Client.
2) Policy setup:
   i) Administrator defines an encryption policy:

Key_Per_Tape

{

Encryption Mode: GCM-128-AES-256

Key Generation Mode: One_Per_Tape

}

   ii) Administrator defines a retention policy:

7_Year_Policy

{

Expiration Time: 30 days after creation

Disable Time: 7 Years, 0 Months, 0 Days

Purge Time: 30 days after disable

}

   iii) Administrator defines a key assignment policy:

Finance_Data_Backup_DataSet

{

Encryption Data Type: Tape Tape Cartridge

Label Range: E0001 – E0010

Usage_Model: Key_Per_Tape

Retention Policy: 7_Year_Policy

}

— Library client operations

3) Writing to new media
   i) Library requests encryption key: GetKeyDataAttribute(Attrib:TapeVolumeId = E0005)
   ii) KMS matches the TapeVolumeID to Finance_Data_Backup_DataSet. KMS generates a GCM-128-AES-256 key and applies both encryption and retentions policies to the newly created key KEY001 and returns it to the client.
   iii) Library uses KEY001 to encrypt volume E0005.
4) Reading from media
   i) Library loads volume and retrieves KeyID KEY001
   ii) Library calls GetKeyById(KEY001)
   iii) KMS checks ACLs and retention policy. If checks pass, KMS sends the key to the client.
   iv) Library uses KEY001 to decrypt the information stored in the volume

B.2 Backup applications

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

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

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

B.3 Laptop/desktop encryption

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

B.4 NAS filer encryption

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

Annex C

(informative)

XML Schema Definitions

[Content TBD]

Annex D

(informative)

Comparison of P1619.3 Key Lifecycle Model with Other Standards

D.1 Key State Comparisons

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

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

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

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

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

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

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

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

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

D.2 Key Transition Comparisons

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

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

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

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

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

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

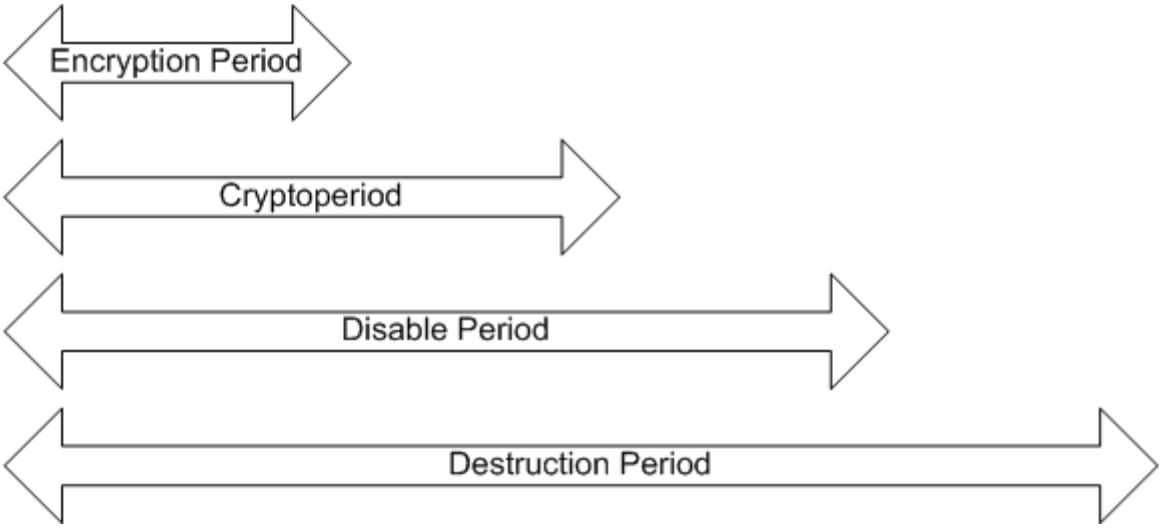
Note 5: IEEE P1619.3 key state transition is implied by the discussion in the NIST SP 800-57 standard, but is not explicitly shown.

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

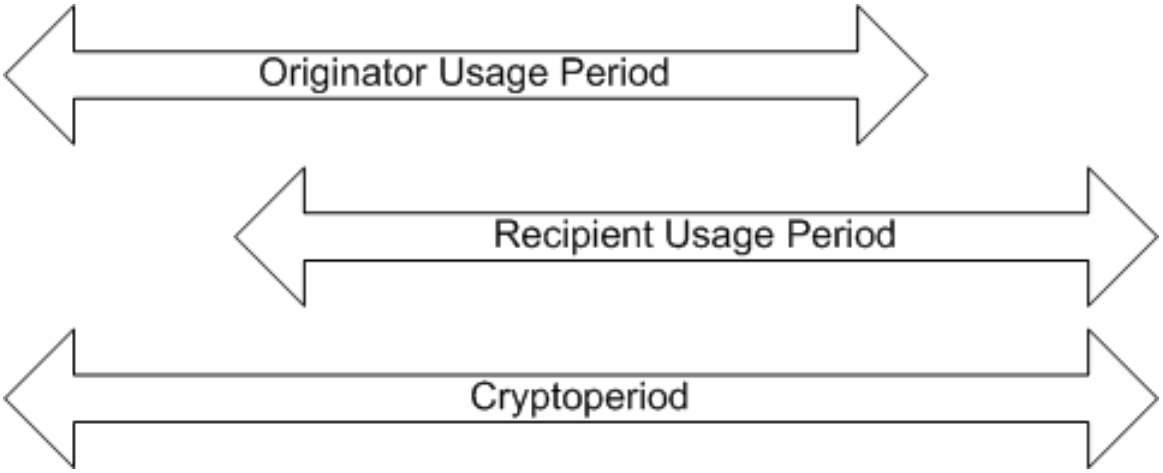
D.3 Key Time Period Comparisons

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

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



D.1 P1619.3 Key Related Time Periods



D.2 NIST SP 800-57 Key Related Time Periods

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

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

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

1

Annex E

(informative)

Discussion of SO_GUID formats

E.1 Comparison Chart of Name Spaces

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

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
Who generates the SO_Handle (KM Client, KMS, KM Client and KMS, KM Client or KMS****)	KM Client or KMS	KM Client or KMS	KM Client or KMS	KM Client	KM Client or KMS

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://-/r/x”

E.2 SO_Family Methods

E.2.1 Required methods

All SO_Families shall document the following methods.

a) form_so_guid:

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

```
typedef unsigned char octet;        // unsigned 8 bit value
typedef short int16_t;  // signed 16 bit value
typedef *octet so_family;           // pointer to 2 octets
typedef {
            octet *value;           // NULL = no value
            int16_t length;         // 0 = no value, < 0 = error code
} so_guid;
typedef {
            octet *value;           // NULL = no value
            int16_t length;         // 0 = no value, < 0 = error code
} so_domain;
typedef {
            octet *value;           // NULL = no value
            int16_t length;         // 0 = no value, < 0 = error code
} so_context;
typedef {
            octet *value;           // NULL = no value
            int16_t length;         // 0 = no value, < 0 = error code
} so_handle;
```

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

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

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

b) parse_so_guid:

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

```
typedef {
            so_family so_family;
            so_guid *so_guid;
            so_domain *so_domain;
            so_handle *so_handle;
} so_parts;
```

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

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

c) is_valid_so_guid:

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

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

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

d) info_so_guid:

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

- When name = “SO_GUID”, value = so_guid data structure pointer
- When name = “SO_GUID.length”, value = the SO_GUID length as an int16_t
- When name = “SO_GUID.value”, value = pointer to the SO_GUID octet array
- When name = “SO_Family”, value = so_family data structure pointer
- When name = “SO_Family.length”, value = the SO_Family length as an int16_t
- When name = “SO_Family.value”, value = pointer to the SO_Family octet array of 2 octets
- When name = “SO_Domain”, value = so_domain data structure pointer
- When name = “SO_Domain.length”, value = the SO_Domain length as an int16_t
- When name = “SO_Domain.value”, value = pointer to the SO_Domain octet array or NULL
- When name = “SO_Context”, value = so_context data structure pointer
- When name = “SO_Context.length”, value = the SO_Context length as an int16_t
- When name = “SO_Context.value”, value = pointer to the SO_Context octet array or NULL
- When name = “SO_Handle”, value = so_handle data structure pointer
- When name = “SO_Handle.length”, value = the SO_Handle length as an int16_t
- When name = “SO_Handle.value”, value = pointer to the SO_Handle octet array
- When name = “is_valid”, value = a bool where true means it was valid and well formed
- [TBD]

```
typedef {
            octet *name;            // NULL = no name
            octet *value;           // NULL = no value
            int16_t n_length;       // 0 = no name, < 0 = error code
```

```
            int16_t v_length;       // 0 = no value, < 0 = error code
} so_data;
typedef {
            so_data *set;           // NULL = no data in set
            int16_t count;          // 0 = empty data set, < 0 = error
} so_info;
```

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.

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.

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

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.

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.

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

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

The SO_Family may document addition methods.

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

E.2.2 km common methods

E.2.2.1 km SO_GUID constructor method

[Content TBD]

E.2.2.2 km SO_GUID parser method

[Content TBD]

E.2.2.3 km valid SO_GUID check method

[Content TBD]

E.2.2.4 km SO_GUID property list method

[Content TBD]

E.2.3 km specific methods

[Content TBD]

E.2.4 na common methods

E.2.4.1 na SO_GUID constructor method

[Content TBD]

E.2.4.2 na SO_GUID parser method

[Content TBD]

E.2.4.3 na valid SO_GUID check method

[Content TBD]

E.2.4.4 na SO_GUID property list method

[Content TBD]

E.2.5 na specific methods

[Content TBD]

E.2.6 rn methods

E.2.6.1 rn SO_GUID constructor method

[Content TBD]

E.2.6.2 rn SO_GUID parser method

[Content TBD]

E.2.6.3 rn SO_GUID valid SO_GUID check method

[Content TBD]

E.2.6.4 rn SO_GUID property list method

[Content TBD]

E.2.7 rn specific methods

[Content TBD]

E.2.8 ek common methods

E.2.8.1 ek SO_GUID constructor method

[Content TBD]

E.2.8.2 ek SO_GUID parser method

[Content TBD]

E.2.8.3 ek valid SO_GUID check method

[Content TBD]

E.2.8.4 ek SO_GUID property list method

[Content TBD]

E.2.9 ek specific methods

[Content TBD]

E.2.10 00 common methods

E.2.10.1 00 SO_GUID constructor method

[Content TBD]

E.2.10.2 00 SO_GUID parser method

[Content TBD]

E.2.10.3 00 valid SO_GUID check method

[Content TBD]

E.2.10.4 00 SO_GUID property list method

[Content TBD]

E.2.11 00 specific methods

[Content TBD]

E.3 SO_Family Advantages, Potential Concerns and Solutions

E.3.1 km SO_Family Advantages, Potential Concerns and Solutions

E.3.1.1 URI Advantages

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

E.3.1.2 URI Potential Concerns and Solutions

- SO_GUID The size is not fixed and can be too large for some implementations

- It is possible to create a standard translation from stored values within specific applications or on media types to a fully qualified SO_GUID or any component of the SO_GUID to allow for retrieval of a key via API calls to a KM Client or within a KM Client that a KM Server will understand.
- While a SO_GUID of the 'km' SO_Family can reach a length of 1024 octets an application can itself enforce the use of a shorter SO_GUID or through translation

E.3.2 na SO_Family Advantages, Potential Concerns and Solutions

E.3.2.1 IEEE OUI Advantages

- Compact format
- Many devices use NAA

E.3.2.2 IEEE OUI Potential Concerns

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

E.3.3 rn SO_Family Advantages, Potential Concerns and Solutions

E.3.3.1 Advantages of Random SO_Handles

- The "rn" SO_Family supports existing naming formats for key identifiers including those that were not chosen at random. Any existing key identifier that is between 1 and 64 octets in length may be used as SO_Handle. Legacy key identifiers may be converted into an "rn" SO_Family SO_GUID by preceding it with the two octets: "rn".
- Random SO_Handles are trivial to generate.
- Random SO_Handles are anonymous. Their value does not directly reveal information about the KMS under which they were first generated.
- Because this SO_Family has no naming authority, using random SO_Handles within a single organization is trivial.

E.3.3.2 Potential Concerns of Random SO_Handles

- Because this SO_Family has no naming authority, exchanging keys between two independent sets of KM Servers is not trivial. The lack of a SO_Domain and SO_Context in the SO_GUID makes is difficult to locate the KM_Servers that have access to the security object. KM_Servers assume that the KMS holds the security object, or they must use information that outside of the scope of this standard to locate the security object or they must assume that the security object does not exist.
- There is no guarantee that a SO_GUID from this SO_Family is globally unique. The chance of two independent entities randomly choosing the same n-octet long SO_Handle (SO_Handle collision) is approximately 256-(n/2).
- Within the same local set of KM_Servers, the KMS is capable of enforcing SO_handle uniqueness by refusing to store two different keys with the same SO_Handle. One solution to the SO_Handle collision problem is to never share keys outside of the local set of KM Servers. However, one cannot assume that a security object will never need to be shared outside the local set of KM Servers. For example, when two independent sets of KM Servers merge (say because two companies merge) there is a potential for SO_Handle collisions. By mapping "rn" SO_Handles

onto different SO_GUIDs of another SO_Family, one may be able to disambiguate the security objects that were previously created in the "rn" SO_Family.

— The SO_Handle collision is a problem between keys of two independent local sets of KM Servers such has may occur when two independent organizations need to share or exchange keys. There is a chance that a SO_Handle from one organization will collide with key from the other organization. Consider the case where a piece of removal media is encrypted by one organization and then is sent to other organization. If the receiving organization has another key with the same SO_Handle in the "rn" SO_Family, them its KM Clients will almost certainly be unable to decrypt the removable media that they received unless they are willing to replace their existing key with the imported key. One solution to this export problem is to convert exported "rn" SO_GUIDs into a SO_GUIDs of SO_Family that supports a global resolvable namespace. For example, if the "example.com" organization wishes to share the key "rn23209" with an external entity, they can export the key as the SO_GUID "km://example.com/-rn/23209".

— Best practices state that for those applications were a proof of SO_GUID uniqueness is not mandatory; a SO_Handle collision of no more than 2-128 is sufficient. Therefore, it is recommended that SO_Handles values contain at least 256 bits of entropy. For example, SO_Handles of at least 32 octets in length that are generated by a cryptographically sound random bit generator would suffice. KM clients that lack the ability to generate such SO_Handles should connect to a capable KM Server and that the KM Server generate the SO_Handle.

E.3.4 ek SO_Family Advantages, Potential Concerns and Solutions

E.3.4.1 Advantages of EKMI Name Space

— Supports existing naming formats for key identifiers

E.3.4.2 Potential Concerns of EKMI Name Space

— No guarantee that identifier is globally unique when keys must be shared between organizations

— Establishment of naming authority for 64 bit

E.3.5 00 SO_Family Advantages, Potential Concerns and Solutions

E.3.5.1 Zero Advantages

— This family has only one valid SO_GUID.

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

E.3.5.2 Zero Potential Concerns and Solutions

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