

To: IEEE P1619.3 Task Group
From: P1619.3 Name Space Subgroup, Members:
Gideon Avida, Decru (NetApp)
Matt Ball, Quantum Corp.
Kevin Butt, IBM
Bob Griffin, RSA Security/EMC
Jack Harwood, RSA Security/EMC
Larry Hofer, Emulex
Joerg Huser, NXP Semiconductor
Glen Jaquette, IBM
Ravi Kavuri, Decru (NetApp)
Bob Lockhart, NeoScale Systems
Kevin Marks, Dell
Luther Martin, Voltage Security
Landon Curt Noll, NeoScale Systems
Arshad Noor, StrongAuth
Chris Williams, Hewlett-Packard

Date: September 5, 2007

Purpose: Proposed changes against P1619.3/D1 to incorporate a global unique key identifier

Introduction

The P1619.3 Name Space Subgroup has been working on a proposal to create a globally unique name space for creating key identifiers. This is an important component for interchange because cryptographic keys may travel between key managers of different organizations, and it needs to be possible to integrate the Key IDs within each manager. A collision of the Key ID in this case can cause many problems, including the loss of one of the keys, or difficulty in resolving the required key.

To create a globally unique Key ID, it is necessary to either use a Naming Authority (such as ICANN or IEEE), or use a sufficiently large random number that the chance of a name space collision is acceptably low (e.g. a 256-bit random number). The group was unable to decide on a single naming authority (or lack thereof), and so we decided to instead create a two-character code that then describes the format of the key identifier, including the naming authority.

This proposal is against P1619.3/D1. New changes are in blue and deletions are in red strikethrough.

Changes to P1619.3/D1

[Remove subclause 4.3.3.1 "Key Object", and replace with following starting at 4.4]

2. Normative References

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 draft 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 Definitions

3.1.1 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.

3.1.2 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.

3.1.3 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).

3.1.4 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).

3.1.5 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.

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

3.1.7 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.

3.1.8 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.

3.1.9 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.

3.1.10 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.

3.1.11 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.

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

3.2 Abbreviations

FQDN	Fully Qualified Domain Name
SO_GUID	security object globally unique identifier
URI	Universal Resource Identifier

4. General overview

4.4 Globally unique identifier for Key Management Services

4.4.1 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

NOTE 1— A name space can chose to make use of all or a portion of a SO_GUID for identifying security objects and shall be defined by the individual name space.

The SO_Family shall be a value from Table A:

Table A — SO_Family values

SO_Family	Description	Name Authority	Ref.
'km'	A format based on a URI name space	ICANN	4.4.2
'na'	A format based on an IEEE OUI name space	IEEE OUI	4.4.3
'rn'	A format based on 32 octet random numbers	None	4.4.4
'ek'	A format based on the EKMI name space	None	4.4.5
'00'	A SO_GUID that does not match any object	None	4.4.6

NOTE 2—All other SO_Family identifiers are reserved for future use. Additions to this standard shall be approved by the IEEE.

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 Section **Error! Reference source not found.** 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 shall 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.

4.4.2 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 B – 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 C 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 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	“i”
PKCS#11	“p” “1” “1”

Security objects with may be referenced in several ways:

- 1) Globally by: SO_Family, SO_Domain, SO_Context and SO_Handle
- 2) SO_Context and SO_Handle within the clients current SO_Domain
- 3) 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/key/Hayward/keyid1
- 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

4.4.3 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 D — Format of a SO_GUID with a family set to 'na'

Bit Octet	7	6	5	4	3	2	1	0
0	'n' (6E ₁₆)							
1	'a' (61 ₁₆)							
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 E— 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.

4.4.4 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.

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

[To be added]

4.4.6 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.

Annex A

(informative)

Discussion of SO_GUID formats

A.1 Comparison Chart of Name Spaces

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

Table A.1 — Attributes of Name Spaces

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

Attributes	URI	IEEE OUI	Random	EKMI	Zero
Who generates the SO_Handle (KM Client, 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

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

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

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

A.2 SO_Family Advantages, Potential Concerns and Solutions

A.2.1 km SO_Family Advantages, Potential Concerns and Solutions

A.2.1.1 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

A.2.1.2 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

A.2.2 na SO_Family Advantages, Potential Concerns and Solutions

A.2.2.1 IEEE OUI Advantages

- Compact format
- Many devices use NAA

A.2.2.2 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.

A.2.3 rn SO_Family Advantages, Potential Concerns and Solutions

A.2.3.1 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.

A.2.3.2 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 it 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 shall use information that outside of the scope of this standard to locate the security object or they shall 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 as 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 removable 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, then 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 where 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.

A.2.4 ek SO_Family Advantages, Potential Concerns and Solutions

A.2.4.1 Advantages of EKMI Name Space

- Supports existing naming formats for key identifiers

A.2.4.2 Potential Concerns of EKMI Name Space

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

A.2.5 00 SO_Family Advantages, Potential Concerns and Solutions

A.2.5.1 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.

A.2.5.2 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.

Annex B

(informative)

Required SO_Family Methods

B.1 SO_Family Methods

B.1.1 Required methods

All SO_Families shall document the following methods.

b) `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*);
```

c) `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* );
```

d) `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.

e) `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.

B.1.2 km common methods

B.1.2.1 km SO_GUID constructor method

[To be added]

B.1.2.2 km SO_GUID parser method

[To be added]

B.1.2.3 km valid SO_GUID check method

[To be added]

B.1.2.4 km SO_GUID property list method

[To be added]

B.1.3 km specific methods

[To be added]

B.1.4 na common methods

B.1.4.1 na SO_GUID constructor method

[To be added]

B.1.4.2 na SO_GUID parser method

[To be added]

B.1.4.3 na valid SO_GUID check method

[To be added]

B.1.4.4 na SO_GUID property list method

[To be added]

B.1.5 na specific methods

[To be added]

B.1.6 rn methods

B.1.6.1 rn SO_GUID constructor method

[To be added]

B.1.6.2 rn SO_GUID parser method

[To be added]

B.1.6.3 rn SO_GUID valid SO_GUID check method

[To be added]

B.1.6.4 rn SO_GUID property list method

[To be added]

B.1.7 rn specific methods

[To be added]

B.1.8 ek common methods

B.1.8.1 ek SO_GUID constructor method

[To be added]

B.1.8.2 ek SO_GUID parser method

[To be added]

B.1.8.3 ek valid SO_GUID check method

[To be added]

B.1.8.4 ek SO_GUID property list method

[To be added]

B.1.9 ek specific methods

[To be added]

B.1.10 00 common methods

B.1.10.1 00 SO_GUID constructor method

[To be added]

B.1.10.2 00 SO_GUID parser method

[To be added]

B.1.10.3 00 valid SO_GUID check method

[To be added]

B.1.10.4 00 SO_GUID property list method

[To be added]

B.1.11 00 specific methods

[To be added]