TRUSTED® COMPUTING GROUP

SPECIFICATION

TCG Storage Feature Set: Media Encryption Key Multiparty Authorization (MEK-MPA)

Version 1.00

April 27, 2026

Contact: admin@trustedcomputinggroup.org

PUBLISHED

DISCLAIMERS, NOTICES, AND LICENSE TERMS

ACKNOWLEDGEMENT

The TCG wishes to thank all those who contributed to this specification. This document builds on work done in various working groups in the TCG and industry at large.

NAME	COMPANY
Jaleel Kazi	Dell, Inc.
Charles Kunzman	Google Inc.
Jeff Anderson	Google Inc.
Joy Shukla	Google Inc.
Srini Narayanamurthy	Google Inc.
Zach Halvorsen	Google Inc.
Frederick Knight	Kioxia Corporation
John Geldman	Kioxia Corporation
Taku Kato	Kioxia Corporation
Alan Arnold	Lenovo Inc.
Alon Cohen	Microchip Technology, Inc.
Chander Kavalipati	Microchip Technology, Inc.
Bharath Madanayakanahalli Gururaj	Micron Technology, Inc.
Robert Strong	Micron Technology, Inc.
Sridhar Balasubramanian	NetApp
Eric Hibbard	Samsung Semiconductor Inc.
Gwangbae Choi	Samsung Semiconductor Inc.
Yoni Shternhell	Sandisk
Anthony Duran	Seagate Technology
Festus Hategekimana	Solidigm
John Mathews	Solidigm
Patrick Hery	Toshiba Corporation
Joseph Chen	ULINK
Jithendra Bethur	Western Digital Technologies
Chandra Nelogal	Invited Expert
Jim Hatfield	Invited Expert

CONTENTS

List of Tables

List of Figures

1 Scope

This specification defines the MEK Multiparty Authorization Feature Set.

1.1 Key Words

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in this document normative statements are to be interpreted as described in RFC-2119, Key words for use in RFCs to Indicate Requirement Levels.

1.2 Statement Type

Please note a very important distinction between different sections of text throughout this document. There are two distinctive kinds of text: informative comment and normative statements. Because most of the text in this specification will be of the kind normative statements, the authors have informally defined it as the default and, as such, have specifically called out text of the kind informative comment. They have done this by flagging the beginning and end of each informative comment and highlighting its text in gray. This means that unless text is specifically marked as of the kind informative comment, it can be considered a kind of normative statement.

Start of informative comment

This is the first paragraph of 1–n paragraphs containing text of the kind *informative comment* ...

This is the second paragraph of text of the kind *informative comment* ...

This is the nth paragraph of text of the kind *informative comment* ...

To understand the TCG specification the user must read the specification. (This use of MUST does not require any action).

End of informative comment

2 Introduction and Concepts

2.1 Document Purpose

The Storage Workgroup specifications provide a comprehensive architecture for Storage Devices (SDs) under policy control as determined by the trusted platform host, the capabilities of the SD to conform to the policies of the trusted platform, and the lifecycle state of the SD as a Trusted Peripheral.

2.2 Intended Audience

The intended audience for this specification is both trusted SD manufacturers and developers that want to use these SDs in their systems.

2.3 Conventions

2.3.1 Font Conventions

Names of methods and Security Provider (SP) tables are in `Courier New font` (e.g., the `Set` method, the `Locking` table). This convention does not apply to method and table names appearing in headings or captions.

Hexadecimal numbers are in `Courier New font`.
All other text is in the Arial font.

2.3.2 Security Provider (SP) Table Cell Symbols Legend

The legend in Table 1 defines the SP table cell symbols used to communicate Read-Write and Access Control attributes of a cell. These symbols are informative only. Unless otherwise indicated, the table cell content is normative. If a table cell is empty or blank (regardless of symbols), then the contents of that cell are not specified in this specification. The contents might be specified in another specification.

Start of Informative Comment

As specified in the TCG Storage Architecture Core Specification (see [2]) section 3.4.2 and section 5.3.4.3, Access Control limits the methods that can be processed on an SP, a table, or on specific rows and columns of a table. Access Control is specified in layers. The top layer of the mechanism is an Access Control List (ACL). ACLs contain lists of Access Control Elements (ACEs). ACEs contain Boolean combinations of authorities. ACLs and ACEs bind methods to the authorities that are permitted to invoke them. When the state of authorities specified in an ACE Boolean expression are satisfied as described in the TCG Storage Architecture Core Specification (see [2]) section 3.4.2.1 (for example, the ACE Boolean expression resolves to True on successful authentication), the ACE permits methods to be invoked on an SP, a table, or on specific rows and columns of a table based on the ACL that contains the ACE.

End of Informative Comment

The “R-W” column in Table 1 describes if the contents of an SP table cell is read-only, writeable, or not required (NR). This cell property can be determined through the mapping of methods, that provide read or write access to the cell, and an ACL in the AccessControl table (see TCG Storage Architecture Core Specification [2] section 5.3.4.3). If a cell is readable, then there exists a method and associated ACE definition to retrieve the contents of the cell. If a cell is writeable, then there exists a method and associated ACE definition to modify the contents of the cell. There MAY be other conditions that prevent a method from being invoked on a table cell outside the scope of the Access Control specified in the AccessControl table.

The “Value” column in Table 1 describes if the contents of an SP table cell is specified by the feature set, vendor unique (VU), or not required (NR). When a cell content is VU, the cell contents contain the text “VU” and access control for the cell is defined by the feature set.

The “Access Control” column in Table 1 describes if the access control associated with an SP table cell is user personalizable, fixed, or not defined (ND). Access Control for a cell is personalizable if the Boolean expression of the ACEs associated with the cell are writable. Access Control for a cell is fixed if the Boolean expression of the ACEs associated with the cell are not writeable. Access Control for a cell is not defined (ND) if there does not exist an ACE that is associated with the cell. If the cell contents are (NR), the `Get` method MAY omit the column from a method response. The behavior of `AddACE` and `RemoveACE` methods with respect to ACLs are not defined.

Table 1 – Security Provider Table Legend

Table Cell Symbols Legend	R-W	Value	Access Control	Comment
Read-only Fixed Access Control (RF)	Read-only	MEK Multiparty Authorization Feature Set specified	Fixed	• Cell content is Read-Only. • Access control is fixed. • Value is specified by the MEK Multiparty Authorization Feature Set
Read-only Fixed Access Control-VU (RFV)	Read-only	VU	Fixed	• Cell content is Read-Only. • Access Control is fixed. • Values are Vendor Unique (VU). A minimum or maximum value might be specified.
Undefined (UD)	Not Defined (ND)	Not Required (NR)	Not Defined (ND)	• Cell content is (NR). • Access control is not defined (ND). • Any text in table cell is informative only. • A `Get` MAY omit this column from the method response.
Writable Personalizable Access Control (WP)	Write	Preconfigured/ user personalizable, or VU	Preconfigured, user personalizable	• Cell content is writable. • Access control is personalizable • `Get` Access Control is not described by this symbol • Cell contents preconfiguration for VU column is “VU”.

Table Cell Symbols Legend	R-W	Value	Access Control	Comment
Writable Fixed Access Control (WF)	Write	Preconfigured/ user personalizable, or VU	Fixed	• Cell content is writable. • Access control is fixed. • `Get` Access Control is not described by this symbol • Cell contents preconfiguration for VU column is “VU”.

2.3.3 List Conventions

2.3.3.1 Lists Overview

Lists are associated with an introductory paragraph or phrase, and are numbered relative to that paragraph or phrase (i.e., all lists begin with an a) or 1) entry).

Each item in a list is preceded by an identification with the style of the identification being determined by whether the list is intended to be an ordered list or an unordered list.

If the item in a list is not a complete sentence, the first word in the item is not capitalized. If the item in a list is a complete sentence, the first word in the item is capitalized.

Each item in a list ends with a semicolon, except the last item, which ends in a period. The next to the last entry in the list ends with a semicolon followed by an “and” or an “or” (i.e., “…; and”, or “…; or”). The “and” is used if all the items in the list are required. The “or” is used if only one or more items in the list are required.

2.3.3.2 Unordered Lists

An unordered list is one in which the order of the listed items is unimportant (i.e., it does not matter where in the list an item occurs as all items have equal importance). In an unordered list, each item starts with a lowercase letter followed by a close parenthesis. If it is necessary to subdivide a list item further with an additional unordered list (i.e., have a nested unordered list), then the nested unordered list is indented and each item in the nested unordered list starts with an uppercase letter followed by a close parenthesis.

The following is an example of an unordered list with a nested unordered list:

EXAMPLE - The following are the items for the assembly:
a) a box containing:
    A) a bolt;
    B) a nut; and
    C) a washer;
b) a screwdriver; and
c) a wrench.

2.3.3.3 Ordered Lists

An ordered list is one in which the order of the listed items is important (i.e., item n is required before item n+1). In an ordered list, each item starts with a Western-Arabic numeral followed by a close parenthesis. If it is necessary to

subdivide a list item further with an additional unordered list (i.e., have a nested unordered list), then the nested unordered list is indented and each item in the nested unordered list starts with an uppercase letter followed by a close parenthesis.

The following is an example of an ordered list with a nested unordered list:

EXAMPLE - The following are the instructions for the assembly:
1) remove the contents from the box;
2) assemble the item;
   A) use a screwdriver to tighten the screws; and
   B) use a wrench to tighten the bolts;
   and
3) take a break.

2.3.4 Numbering Conventions

A binary number is represented in this specification by any sequence of digits consisting of only the Western-Arabic numerals 0 and 1 immediately followed by a lowercase b (e.g., 0101b). Underscores or spaces may be included between characters in binary number representations to increase readability or delineate field boundaries (e.g., 0 0101 1010b or 0_0101_1010b).

A hexadecimal number is represented in this specification by any sequence of digits consisting of only the Western-Arabic numerals 0 through 9 and/or the uppercase English letters A through F immediately preceded by "`0x`". Underscores or spaces might be included between characters in hexadecimal number representations to increase readability or delineate field boundaries (e.g., `0xFD8C FA23` or `0x0B_FD8C_FA23`). Hexadecimal numbers are in `Courier New font`.

A decimal number is represented in this standard by any sequence of digits consisting of only the Western-Arabic numerals 0 through 9 not immediately followed by a lowercase b or lowercase h (e.g., 25). This standard uses the following conventions for representing decimal numbers:

a) the decimal separator (i.e., separating the integer and fractional portions of the number) is a period;
b) the thousands separator (i.e., separating groups of three digits in a portion of the number) is a space; and
c) the thousands separator is used in both the integer portion and the fraction portion of a number.

A decimal number represented in this standard with an overline over one or more digits following the decimal point is a number where the overlined digits are infinitely repeating (e.g., $666.\overline{6}$ means 666.666 666… or 666 2/3, and $12.\overline{142857}$ means 12.142 857 142 857… or 12 1/7).

2.3.5 Bit Conventions

Name (n:m), where n is greater than m, denotes a set of bits (e.g., Feature (7:0)).

2.3.6 Number Range Conventions

p..q, where p is less than q, represents a range of numbers (e.g., words 100..103 represents words 100, 101, 102, and 103).

2.3.7 Specify and Indicate Conventions

In a given message, command, or other information exchange between a requestor (e.g., a host) and a responder (e.g., an SD):

a) the requestor specifies information in the request; and
b) the responder indicates information in the response.

2.4 Document References

2.4.1 Document Precedence

If there is a conflict between this specification and any other reference, then the precedence is (where a lower number indicates higher precedence):

1. This specification;
2. Normative references (see section 2.4.2).

Each normative reference and informative reference might specify its own document precedence.

2.4.2 Normative References

[1] IETF RFC 2119, 1997, “Key words for use in RFCs to Indicate Requirement Levels”

[2] Trusted Computing Group (TCG), “TCG Storage Architecture Core Specification”, Version 2.01

[3] Trusted Computing Group (TCG), “TCG Storage Security Subsystem Class: Opal”, Version 2.30

[4] IETF RFC 5280, 2008, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile”

[5] IETF RFC 6234, 2011 “US Secure Hash Algorithms”

[6] IETF RFC 9180, 2022, “Hybrid Public Key Encryption”

[7] Trusted Computing Group (TCG), “Implicit Identity Based Device Attestation”, Version 1.0

[8] Trusted Computing Group (TCG), “TCG Storage Opal SSC Feature Set: Single User Mode“, Version 1.00

[9] Trusted Computing Group (TCG), “TCG Storage Feature Set: Configurable Locking for NVMe Namespaces and SCSI LUNs “, Version 1.02.

[10] Trusted Computing Group (TCG), “TCG Storage Interface Interactions Specification”, Version 1.11

[11] NIST, SP800-57, 2020, “Recommendations for Key Management: Part 1 – General”. Part1, Rev.5

2.4.3 Informative References

This feature set does not include any informative references.

2.5 Dependencies on Other Feature Sets

This feature set does not depend upon any other feature sets.

2.6 Interactions with Other Feature Sets

This feature set defines the interactions with the Single User Mode Feature Set (see [8]), as described in 5.7.

This feature set defines the interactions with the Configurable Locking for NVMe Namespaces and SCSI LUNs Feature Set (see [9]), as described in 5.8.

2.7 Definition of Terms

Term	Definition
AccessCondition	An AccessCondition object (see 4.2.1.1) represents sealed cryptographic material, where the sealed cryptographic material is used in combination with other cryptographic material to seal access (when configured) to one or more media encryption keys
AccessEnabled	The state of an AccessCondition object with an AccessEnabled column value of True (see 4.2.1.1.4)
access_key	A host-owned, externally supplied cryptographically random value that is used to control the state of each AccessCondition object (for example, it is used to unseal the cryptographic material in an AccessCondition object to transition it to AccessEnabled state (see 4.1.1.4)).
Current access_key	The access_key value most recently configured by a successful `InitializeAccessCondition` or `ChangeAccessKey` method call on an object.
Eradicate	Irrevocably erase (e.g., cryptographically erase)
IF-RECV	An interface command used to retrieve security protocol data from the TPer (see [2])
IF-SEND	An interface command used to transmit security protocol data to the TPer (see [2])
KEM	A Key Encapsulation Mechanism (KEM) is a cryptographic design pattern for sending data to a receiver using a KEM public key to encapsulate a shared secret, which is used to derive an encryption key for a symmetric encryption algorithm that encrypts data sent to the TPer. See [6] for more details.
Locking SP	A security provider that incorporates the Locking Template as described in the Core Spec (see [2])
Locking SP is owned	A condition in which specific modifications of an SP have been made (see [10])
ND	Not Defined.
NR	Not Required.
Opal Family	Any of: Opal SSC, Opalite SSC, Ruby SSC, or Pyrite SSC
Read Locked	The state of a Locking object with a ReadLockEnabled column value of True and a ReadLocked column value of True (see [2])
Read Unlocked	The state of a Locking object with a ReadLockEnabled column value of False or a ReadLocked column value of False (see [2])
SD	Storage Device

Term	Definition
SP	Security Provider
SSC	Security Subsystem Class. SSC specifications describe profiled sets of TCG functionality
TPer	The TCG security subsystem within a Storage Device
Write Locked	The state of a Locking object with a WriteLockEnabled column value of True and a WriteLocked column value of True (see [2])
Write Unlocked	The state of a Locking object with a WriteLockEnabled column value of False or a WriteLocked column value of False (see [2])

3 MEK Multiparty Authorization Overview

Start of Informative Comment

The MEK Multiparty Authorization (MEK MPA) feature set enables joint access control to a TPer's media encryption keys (MEKs) managed by `K_AES_256` table objects, ensuring that multiple distinct authorities can collectively authorize access in a secure manner. This capability is useful in multi-layered environments like cloud computing, where data security responsibilities may be distributed among Cloud Service Providers (CSPs) and their tenants.

The TCG Opal specification and associated features sets support user-defined access control over Locking ranges through mechanisms like ReadLocked, WriteLocked, ReadLockEnabled, and WriteLockEnabled columns. However, these mechanisms permit single or multiple users to act unilaterally, lacking native support for joint decision-making between distinct authorities acting at different points in time. Existing alternatives, such as layered encryption, introduce significant trust dependencies or cost and processing overheads.

The MEK MPA feature set addresses this by introducing AccessCondition objects that augment existing user-defined C_PIN-based access control architecture (see section 5 of [2]) to implement an asynchronous joint access control to a TPer's MEKs. Each AccessCondition object state is controlled by an access_key, which is a cryptographically random bit string provided by a host entity to a TPer at runtime, protected in transit using Hybrid Public-Key Encryption (HPKE [6]), and ephemerally-stored by the TPer.

The feature set enforces that all AccessCondition objects are cryptographically associated with a given Locking object's K_AES_256 MEK(s) such that any removal or modification of the AccessCondition objects associated with K_AES_256 object renders the K_AES_256 MEK(s) irrecoverable. In addition, the feature set also enforces that all AccessCondition objects associated with a given Locking object's K_AES_256 MEK(s) need to be enabled, in conjunction with other user-defined C_PIN-based access control, before the corresponding K_AES_256 MEK(s) object can be accessed or utilized, enabling true multiparty control.

Figure 1 illustrates the resulting access control model for accessing a Locking object's K_AES_256 MEK(s).

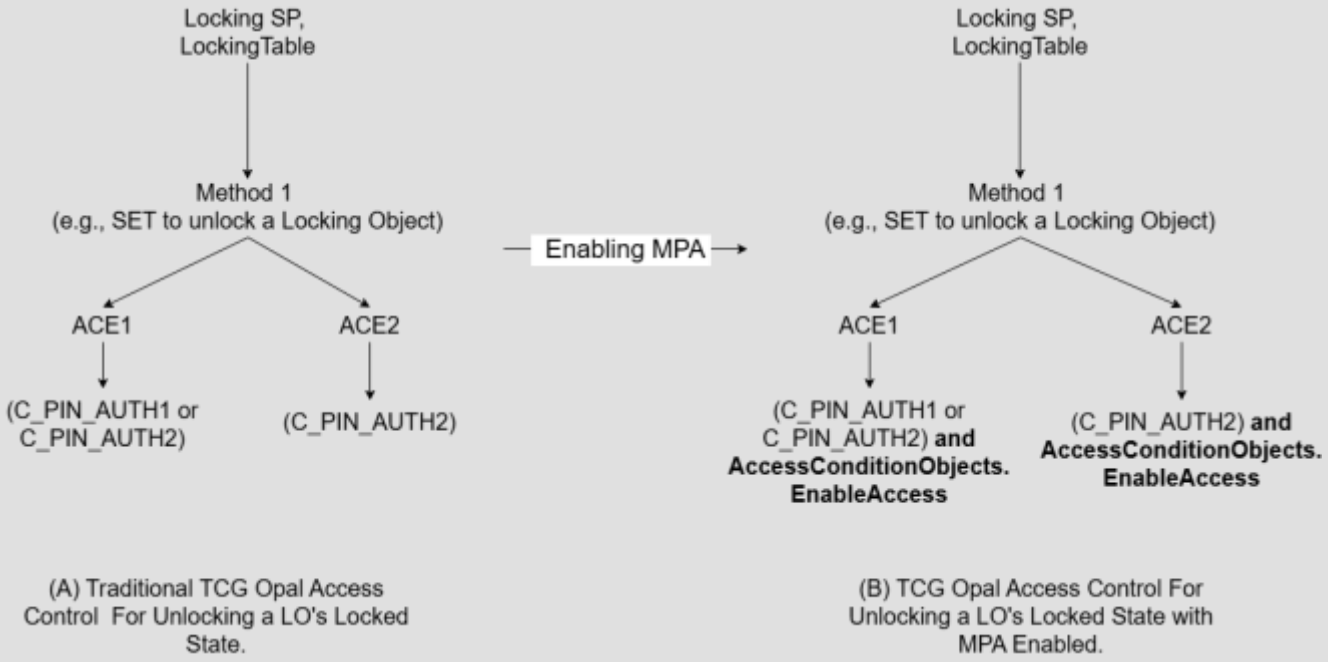

Figure 1 MEK Multiparty Authorization Scheme

Key Features:

- Feature Discoverability: A feature descriptor in Level 0 Discovery response data to allow a host to discover whether a TPer supports MEK MPA feature and the supported MPA capabilities.
- New Object Tables:
    - `AccessCondition` table which lists `AccessCondition` objects supported by a TPer, enabling their unique identifiers discovery and association with one or more K_AES_256 objects. The runtime state of each AccessCondition object is binary: enabled (authorized via access_key) or disabled (revoked or Storage Device reset). These objects allow granular control over MEKs access without affecting TCG Opal's core behavior when not configured.
    - `Certificates` table which stores a TPer's endorsed KEM public key certificate that is used by a host to securely provide an encrypted access_key to a TPer via a Hybrid Public Key Encryption (see [6]) scheme. The access_key serves to authenticate host access to one or more AccessCondition objects.
- New TCG Opal methods to allow a host to manage the lifecycle of AccessCondition objects within the TPer.
- TCG Opal methods modifications to allow a host to request a TPer to bind one or more AccessCondition objects with a TPer's K_AES_256 MEK(s) using existing TCG Opal methods.
- Immutable Cryptographic Sealing: access_key cryptographically seals the AccessCondition. The latter is then used to cryptographically seal the K_AES_256 MEK.
- Compatibility with Single User Mode (SUM): MPA expands the access model by enabling multi-authority control while preserving compatibility with SUM's mechanisms for managing singular user control of Locking ranges.
- Compatibility with Configurable Namespace Locking (CNL): AccessConditions integrate seamlessly with namespace locking, enabling secure partitioning of responsibilities while supporting dynamic reconfiguration.
- Backward Compatibility: If AccessConditions are not explicitly configured, the MEK MPA feature set introduces no changes to the behavior defined in TCG Opal's base specification or other feature sets.

By ensuring that AccessCondition' access_key is only ephemerally stored in the Storage Device and is lost on device resets (see section 4.2.1.1.5), this feature set provides an enhanced Data-At-Rest protection architecture for stored user data against unauthorized access once it leaves the owner's control (e.g., following a power cycle and subsequent de-authentication).

The feature set also ensures operational efficiency, enabling scalable and cost-effective implementation of multiparty authorization workflows within TCG Opal-compliant systems.

End of Informative Comment

4 Security Subsystem Class (SSC) Specific Functionality

This section specifies the additional SSC-specific functionality (not contained in [3]) required to support this MEK Multiparty Authorization Feature Set.

4.1 Methods

This section defines new methods and modifications to existing methods required for this feature set.

4.1.1 New Methods

This section defines the new methods that are required to support this feature set.

4.1.1.1 InitializeAccessCondition

Start of Informative Comment

The `InitializeAccessCondition` method initializes the state of the AccessCondition object and assigns the encrypted_access_key provided to the AccessCondition object.

In addition, this method enables a host to recover its ability to manage the lifecycle of an AccessCondition object in the event the associated access_key is lost, and a new one needs to be provisioned.

Due to the first-come first-served model of the `InitializeAccessCondition`, the AccessCondition object can't be configured in any table or be associated with any K_AES_256 MEK at the time this method is invoked, as doing otherwise would effectively make `InitializeAccessCondition` a cryptographic erase operation for all data protected by media encryption keys which are sealed by the AccessCondition.

Additionally, the AccessCondition object is expected to be in disabled state at the time this method is invoked to prevent unauthorized change in the ownership of that AccessCondition object.

End of Informative Comment

The `InitializeAccessCondition` method is a Locking Template-specific method.

The following pseudo-code is the signature of the `InitializeAccessCondition` method (see 4.1.1.1.3 for more information).

```
AccessConditionObjectUID.InitializeAccessCondition [
    encapsulated_key : KEM_encapsulated_key,
    encrypted_access_key : KEM_encapsulated_data{sequence_number = 0, access_key} ]
=>
[ ]
```

Method UID: 00 00 00 06 00 00 0B 01

4.1.1.1.1 Parameter Descriptions

4.1.1.1.1.1 encapsulated_key

The encapsulated_key parameter specifies the encapsulation (enc) output of the Key Encapsulation Mechanism (KEM) Encap() function, as defined in [6]. This enc value is used in conjunction with the sender's ephemeral private key (sk_S) to derive the shared secret (shared_secret) through the KEM Decap() function. The shared_secret is then used as input to the Key Schedule, which ultimately produces the encryption key for the encrypted_access_key parameter value.

For this method, the EncapsulationKey of the AccessCondition object (see 4.2.1.1.6) specifies the certificate object used for decapsulation. If the encapsulated_key value cannot be decapsulated using the private key associated with the certificate object, the method SHALL fail with a status of INVALID_PARAMETER.

When deriving the key needed to decrypt the encrypted_access_key parameter, the info argument that is passed to SetupBaseR() from section 5.1.1 of [6] SHALL be set to the concatenation of the hex value of the `InitializeAccessCondition` method UID and the hex value of the target AccessCondition object UID.

4.1.1.1.1.2 encrypted_access_key

The encrypted_access_key parameter specifies a KEM_encapsulated_data (see 4.3.1.1.2) value, where the encrypted data is the access_key that MUST be associated with the AccessCondition object after initialization. The sequence_number indicates the sequence of the encryption operation performed using HPKE (see 4.3.1.1.2).

If the KEM_encapsulated_data is not decryptable using the key derived from the encapsulated_key parameter, then the method SHALL fail with a status of INVALID_PARAMETER.

4.1.1.1.2 Returned Value Descriptions

The `InitializeAccessCondition` method returns no value.

4.1.1.1.3 InitializeAccessCondition Method Operation

The operation of the `InitializeAccessCondition` method is to reinitialize the state of the AccessCondition object and to bind the decrypted access_key provided to the AccessCondition object. The TPer SHALL ensure that the AccessCondition object is not configured in any table (see section 4.2.2.2) or associated with any key at the time this method is invoked.

If the target AccessCondition object is associated with a K_AES_256 MEK object, then the method SHALL fail with a status of INVALID_PARAMETER.

If the target AccessCondition object does not have an AccessEnabled column value of False, then the method SHALL fail with a status of FAIL.

The access_key provided by the host SHALL NOT be stored persistently in any form on the TPer.

Start of Informative Comment

Since encrypted access_key values are used as encryption keys as part of cryptographic operations, they should be cryptographically random values. And since these are host-provisioned values, held external to the TPer, it's up to the host to ensure their quality of randomness.

The length of the access_key supported by the TPer, as reported in Level 0 Discovery (see section 5.2), needs to be a key size sufficient for strong encryption (see [11] for key sizes recommendations).

End of Informative Comment

4.1.1.2 TestAccessKey

The `TestAccessKey` method is a Locking Template-specific method.

The following pseudo-code is the signature of the `TestAccessKey` method.

```
AccessConditionObjectUID.TestAccessKey [
    encapsulated_key : KEM_encapsulated_key,
    current_encrypted_access_key : KEM_encapsulated_data{sequence_number = 0, access_key},
    nonce : bytes_32 ]
=>
[ result : bytes_48 ]
```

Method UID: 00 00 00 06 00 00 0B 02

4.1.1.2.1 Parameter Descriptions

4.1.1.2.1.1 encapsulated_key

The encapsulated_key parameter specifies the encapsulated_key output of the Key Encapsulation Mechanism (KEM) Encap function that was used to generate the shared secret used to encrypt the current_encrypted_access_key and parameter value.

If the encapsulated_key value is not able to be decapsulated by the private key associated with the certificate object in the EncapsulationKey of the target AccessCondition object (see 4.2.1.1.6), then the method SHALL fail with a status of INVALID_PARAMETER.

When deriving the key needed to decrypt the current_encrypted_access_key parameter, the info argument passed to SetupBaseR() from section 5.1.1 of [6] SHALL be set to the concatenation of the hex value of the `TestAccessKey` method UID and the hex value of the target AccessCondition object UID.

4.1.1.2.1.2 current_encrypted_access_key

The current_encrypted_access_key parameter specifies a KEM_encapsulated_data (see 4.3.1.1.2) value, where the encrypted data is the current access_key for the AccessCondition object. The sequence_number indicates the sequence of the encryption operation performed using HPKE (see 4.3.1.1.2).

If the KEM_encapsulated_data is not decryptable using the key derived from the encapsulated_key parameter, then the method SHALL fail with a status of INVALID_PARAMETER.

If the decrypted access_key value does not match the currently configured access_key for the AccessCondition object, then the method SHALL fail with a status of FAIL.

4.1.1.2.1.3 nonce

The nonce parameter specifies a random value provided by the host to be included in the hash calculation for the method return value.

4.1.1.2.2 Returned Value Descriptions

The `TestAccessKey` method SHALL return the SHA2-384 hash of the concatenation of the following values in the order they are listed, where concatenation's first entry (i.e., the AccessCondition object UID) occupies the most-significant bits position and the last entry (i.e., the nonce) occupies the least-significant bits position:

1. AccessCondition object UID;
2. the currently configured access_key value; and
3. the nonce parameter

4.1.1.2.3 TestAccessKey Method Operation

Start of Informative Comment

The `TestAccessKey` method is designed to allow verification of the configured access_key without affecting the current state of the AccessCondition. This ensures that the method can be used for testing purposes without inadvertently changing the access or locking state of the associated ranges.

Since there is a risk that an authority credential, such as a C_PIN, might be compromised, there exists a race condition where an attacker may use the compromised credential to inject their own encrypted_access_key value using the `InitializeAccessCondition` method after the intended value is set, but before a target AccessCondition object is configured for use with any K_AES_256 MEK objects. This method provides a way for the encrypted_access_key owners to validate that the intended encrypted_access_key is active after configuration, but before writing data to the protected range.

The use of a nonce in this method serves to prevent response replay attacks. By including a fresh nonce with each request, the method ensures that an attacker cannot simply replay a previous successful response to trick the host into believing that a particular access_key is still in place when it may have been changed.

The failure of the method does not result in any configuration changes for the specified AccessCondition object. The host is expected to reinitialize or change the access_key associated with the AccessCondition object.

End of Informative Comment

The purpose of the `TestAccessKey` method is to validate whether a given encrypted access_key value is the current_encrypted_access_key value configured for the AccessCondition object. The method SHALL return the SHA2-384 hash value (see [5]) defined in 4.1.1.2.2 if the encapsulated encrypted access_key value matches the currently configured access_key for the target AccessCondition object. Otherwise, the method SHALL fail with a status of FAIL.

The access_key provided by the host SHALL NOT be stored persistently in any form on the TPer.

4.1.1.3 ChangeAccessKey

Start of Informative Comment

The `ChangeAccessKey` method allows the host to rotate the access_key value associated with a target AccessCondition object. To prevent unauthorized usage, changing access_key values requires proof of possession for the current access_key value.

This is enforced by requiring that the new access_key value is encrypted under the same HPKE context as the current access_key value, with incrementing sequence_number (see section 4.3.1.1.2 for more details).

Only the host entity which knows the current access_key value can create a current_encrypted_access_key payload that can be successfully decrypted by the TPer, and only the creator of the first payload can create the next payload in the sequence, the new_encrypted_access_key, using the same context.

End of Informative Comment

The `ChangeAccessKey` method is a Locking Template-specific method.

The following pseudo-code is the signature of the `ChangeAccessKey` method.

```
AccessConditionObjectUID.ChangeAccessKey [
    encrypted_access_key_encapsulated_key : KEM_encapsulated_key,
    current_encrypted_access_key : KEM_encapsulated_data{sequence_number = 0, access_key,
    new_encrypted_access_key : KEM_encapsulated_data{sequence_number = 1, access_key}]
=>
[ ]
```

Method UID: 00 00 00 06 00 00 0B 03

4.1.1.3.1 Parameter Descriptions

4.1.1.3.1.1 encrypted_access_key_encapsulated_key

The encrypted_access_key_encapsulated_key parameter specifies the encapsulated_key output of the Key Encapsulation Mechanism (KEM) Encap function that was used to generate the shared secret used to encrypt the current_encrypted_access_key and the new_encrypted_access_key.

If the encrypted_access_key_encapsulated_key value is not able to be decapsulated by the private key associated with the certificate object in the EncapsulationKey column of the target AccessCondition object (see section 4.2.1.1.6), then the method SHALL fail with a status of INVALID_PARAMETER.

When deriving the key needed to decrypt the current_encrypted_access_key parameter and the new_encrypted_access_key parameters, the info argument passed to SetupBaseR() from section 5.1.1 of [6] SHALL be set to the concatenation of the hex value of the `ChangeAccessKey` method UID and the hex value of the target AccessCondition object UID.

4.1.1.3.1.2 current_encrypted_access_key

The current_encrypted_access_key parameter specifies a KEM_encapsulated_data (see 4.3.1.1.2) value, where the encrypted data is the current access_key for the AccessCondition object. The sequence_number indicates the sequence of the encryption operation performed using HPKE (see 4.3.1.1.2)..

The current_encrypted_access_key and the new_encrypted_access_key parameters are expected to be encrypted under the same HPKE context with incrementing sequence_number (see [6] for additional details).

As the method signature definition in section 4.1.1.3 indicates, the current_encrypted_access_key is expected to be the first message encrypted under the HPKE context with sequence_number initially at 0.

The TPer SHALL decrypt the current_encrypted_access_key and the new_encrypted_access_key parameters using the same HPKE context. The TPer SHALL perform decryption in that order using ContextR.Open() as defined in section 4.3.1.1.2.

Start of Informative Comment

Note that each invocation of ContextR.Open() will automatically increment the sequence number by 1 between decrypting the current_encrypted_access_key and the new_encrypted_access_key parameter.

End of Informative Comment

If the KEM_encapsulated_data is not decryptable using the key derived from the encrypted_access_key_encapsulated_key parameter, then the method SHALL fail with a status of INVALID_PARAMETER.

If the decrypted access_key value does not match the currently configured access_key for the target AccessCondition object, then the method SHALL fail with a status of FAIL.

4.1.1.3.1.3 new_encrypted_access_key

The new_encrypted_access_key parameter specifies KEM_encapsulated_data (see 4.3.1.1.2) value, where the encrypted data is the new access_key value to be associated with the target AccessCondition object. The sequence_number indicates the sequence of the encryption operation performed using HPKE (see 4.3.1.1.2).

As defined in section 4.1.1.3.1.2, the current_encrypted_access_key and the new_encrypted_access_key parameters are expected to be encrypted under the same HPKE context with incrementing sequence_number and SHALL be decrypted by the TPer in that same order with incrementing sequence_number.

If the KEM_encapsulated_data is not decryptable using the key derived from the encrypted_access_key_encapsulated_key parameter, then the method SHALL fail with a status of INVALID_PARAMETER.

4.1.1.3.2 Returned Value Descriptions

The `ChangeAccessKey` method returns no values.

4.1.1.3.3 ChangeAccessKey Method Operation

Start of Informative Comment

The AccessEnabled and Locked states of the associated K_AES_256 MEK object does not change as a result of this method. The `ChangeAccessKey` method only updates the encrypted_access_key value and does not affect the current state of access or locking.

Since the current encrypted_access_key is required to change the encrypted_access_key for the AccessCondition object, losing the currently configured encrypted_access_key will render the AccessCondition object unusable once disabled. In that situation, the `InitializeAccessCondition` method can be used to reset the AccessCondition object, which results in loss of any associated data.

End of Informative Comment

The TPer SHALL ensure that for the target AccessCondition object, the operation of the `ChangeAccessKey` method changes only the access_key value. After returning successful status for the method, the TPer SHALL ensure that future invocations of the `EnableAccess` and `ChangeAccessKey` methods will only accept the new access_key value as the encrypted data in the encrypted_access_key parameter.

The access_key values provided by the host SHALL NOT be stored persistently in any form on the TPer

4.1.1.4 EnableAccess

The `EnableAccess` method is a Locking Template-specific method.

The following pseudo-code is the signature of the `EnableAccess` method.

```
AccessConditionObjectUID.EnableAccess [
    encapsulated_key : KEM_encapsulated_key,
    current_encrypted_access_key : KEM_encapsulated_data{sequence_number = 0, access_key} ]
=>
[ ]
```

Method UID: 00 00 00 06 00 00 0B 04

4.1.1.4.1 Parameter Descriptions

4.1.1.4.1.1 encapsulated_key

The encapsulated_key parameter specifies the encapsulated_key output of the Key Encapsulation Mechanism (KEM) Encap function that was used to generate the Shared Secret used to encrypt the encrypted_access_key parameter value.

If the encapsulated_key value is not able to be decapsulated by the private key associated with the certificate object in the EncapsulationKey column of the target AccessCondition object (see 4.2.1.1.6), then the method SHALL fail with a status of INVALID_PARAMETER.

When deriving the key needed to decrypt the encrypted access_key parameter, the info argument passed to SetupBaseR() from section 5.1.1 of [6] SHALL be set to the concatenation of the hex value of the `EnableAccess` method UID and the hex value of the target AccessCondition object UID.

4.1.1.4.1.2 current_encrypted_access_key

The current_encrypted_access_key parameter specifies a KEM_encapsulated_data (see 4.3.1.1.2) value, where the encrypted data is the current access_key for the AccessCondition object. The sequence_number indicates the sequence of the encryption operation performed using HPKE (see 4.3.1.1.2).

If the KEM_encapsulated_data is not decryptable using the key derived from the encapsulated_key parameter, then the method SHALL fail with a status of INVALID_PARAMETER.

If the decrypted access_key value does not match the currently configured access_key for the AccessCondition object, then the method SHALL fail with a status of FAIL.

4.1.1.4.2 Returned Value Descriptions

The `EnableAccess` method returns no values.

4.1.1.4.3 EnableAccess Method Operation

If the encrypted_access_key parameter provided to the `EnableAccess` method, once decrypted, matches the current access_key, the TPer SHALL set the value of the AccessEnabled column of the AccessCondition object to True.

If the decrypted access_key parameter does not match the currently configured access_key, the TPer SHALL fail the method with a status of INVALID_PARAMETER.

If the `EnableAccess` method is invoked on an AccessCondition object that has not been initialized, the TPer SHALL fail the method with a status of INVALID_PARAMETER.

The TPer SHALL NOT change the ReadLocked or WriteLocked states of any associated Locking object in response to a successful method call.

For details on the restrictions imposed by the ActiveAccessConditions column, refer to the `K_AES_256` table modifications section of this feature set (i.e., section 4.2.2.2).

The access_key provided by the host SHALL NOT be stored persistently in any form on the TPer.

4.1.1.5 DisableAccess

The `DisableAccess` method is a Locking Template-specific method.

The following pseudo-code is the signature of the `DisableAccess` method (see 4.1.1.5.3 for more information).

```
AccessConditionObjectUID.DisableAccess [ ]
=>
[ ]
```

Method UID: 00 00 00 06 00 00 0B 05

4.1.1.5.1 Parameter Descriptions

The `DisableAccess` method has no parameters.

4.1.1.5.2 Returned Value Descriptions

The `DisableAccess` method returns no values.

4.1.1.5.3 DisableAccess Method Operation

The operation of the `DisableAccess` method is to change the value of the AccessEnabled column of the AccessCondition object to False[.

If:

a) the AccessCondition object is present in any `K_AES_256` table object's ActiveAccessConditions column value; and

b) the AccessEnabled column of the AccessCondition object is set to False,

then:

a) The media encryption key of that `K_AES_256` table object SHALL become inaccessible; and

b) The TPer SHALL transition the associated Locking object in a read-locked and write-locked state and restrict access to Read, Write and Cryptographic erase operations as specified in [10].

The behaviors described above SHALL remain in effect until either the AccessEnabled column is set to True[(see section 4.2.1.1.4) or the AccessCondition object reference is removed from the `K_AES_256` table object's ActiveAccessConditions column value.

Start of Informative Comment

As noted in the normative text of `EnableAccess`, a Locking object will not automatically transition back to the readable or writable state when an associated AccessCondition object is enabled. It needs to be explicitly transitioned using the Set method once all AccessCondition objects in the ActiveAccessConditions list of the `K_AES_256` table object are enabled.

Removing a disabled AccessCondition object reference from the ActiveAccessConditions list of the `K_AES_256` table object can be done through multiple methods, including:

- `GenKey` (see section 4.1.2.5)
- `Revert` (see section 4.1.2.2)
- `RevertSP` (see section 4.1.2.3)
- `Erase` (see section 5.7.2.2)

Because invoking `InitializeAccessCondition` requires the AccessCondition object to be disabled (see Section 4.1.1.1 for preconditions and Section 4.1.1.5.3 for the definition of “disabled” state), and that method is the only recovery method for a lost access_key, the `DisableAccess` method does not require the current encrypted_access_key as a precondition to disabling access. Instead, access control to this method is solely established by the ACE configuration for the method (see Section 5.3.4 of [2]).

This intentionally allows CSPs (Cloud Service Providers) to disable access to data when, for example, a tenant VM (Virtual Machine) terminates and the TPer was using a configuration that gave the tenant an AccessCondition object to restrict access to their data. This ensures that the data is only accessible once the tenant affirmatively enables the AccessCondition object at some later point in time.

End of Informative Comment

4.1.2 Modified Methods

This feature set modifies the following methods:

a) `Activate`;

b) `Revert`;

c) `RevertSP`;

d) `Set`; and

e) `GenKey`

4.1.2.1 Activate

When the `Activate` method completes successfully, objects in the `Certificates` table where the EphemeralKeyPair column is True SHALL be modified as specified in section 4.2.2.1.1.

Start of Informative Comment

The behavioral requirements of Activate as described in [3] are fully compatible with this feature set. Prior to activation, no media encryption keys will be bound to any `AccessCondition` objects, and therefore requirements around them do not interact with the behavioral restrictions from the `AccessCondition` objects that are defined in this feature set.

End of Informative Comment

4.1.2.2 Revert

Start of Informative Comment

This section clarifies ordering constraints for the `Revert` method processing relative to the ActiveAccessConditions and PendingAccessConditions columns of the `K_AES_256` table. Per section 5.1.2.2 of [3], if the Locking SP is not in the Manufacturing-Inactive state, then successful invocation of the Revert method on either the Admin SP or Locking SP requires the TPer to eradicate all media encryption keys in addition to the ActiveDataRemovalMechanism (section 4.2.7.1 of [3]).

End of Informative Comment

When performing the Cryptographic Erase portion of the `Revert` method, any new media encryption keys SHALL be generated after the ActiveAccessConditions and PendingAccessConditions columns are reverted to the empty list values as specified in 4.2.2.2.1 and 4.2.2.2.2.

Any ephemeral copies of data associated with AccessCondition objects, including encrypted access_keys SHALL be eradicated from volatile memory.

4.1.2.3 RevertSP

Start of Informative Comment

This section clarifies ordering constraints for the `RevertSP` method processing relative to the ActiveAccessConditions and PendingAccessConditions columns of the `K_AES_256` table. Per section 5.1.3.3 of [3], the successful invocation of the RevertSP method on the Locking SP requires the TPer to eradicate all media encryption keys in addition to the ActiveDataRemovalMechanism (section 4.2.7.1 of [3]).

End of Informative Comment

When performing the Cryptographic Erase portion of the `RevertSP` method, the new media encryption keys SHALL be generated after the ActiveAccessConditions and PendingAccessConditions columns are reverted to the empty list values as specified in 4.2.2.2.1 and 4.2.2.2.2.

Any ephemeral copies of data associated with AccessCondition objects, including encrypted access_keys SHALL be eradicated from volatile memory.

Start of Informative Comment

This clause defines constraints on using the KeepGlobalRangeKey parameter of the `RevertSP` when the MEK Multiparty Authorization feature set is used with the Global Range `Locking` object.

End of Informative Comment

If:

- the `RevertSP` method is invoked on the LockingSP;
- the `K_AES_256` table object for the Global Range has a non-empty ActiveAccessConditions column value; and
- the KeepGlobalRangeKey parameter is set to True

Then the method SHALL fail with a status of INVALID_PARAMETER.

Start of Informative Comment

KeepGlobalRangeKey method parameter being set to True requires the media encryption key to be preserved across the LockingSP state transition. However, the LockingSP is required to otherwise revert to the Original Factory State, including empty ActiveAccessConditions column values as specified in the `K_AES_256` table preconfiguration changes in this feature set.

This feature set intentionally does not allow rebinding of media encryption keys to different AccessCondition object sets to enable stronger security statements about media encryption keys dependencies on encrypted_access_keys across their entire lifetime. Therefore, this feature set requires that the Global Range be explicitly unbound from all AccessCondition objects using `GenKey` prior to any `RevertSP` method call that uses the KeepGlobalRangeKey parameter with a True value.

End of Informative Comment

4.1.2.4 Set

When a `Set` method targets the ActiveAccessConditions column of a `K_AES_256` table object, the method behavior is modified as described in 4.2.2.2.1.

When a `Set` method targets the ReadLockEnabled, WriteLockEnabled, ReadLocked, or WriteLocked columns of a `Locking` table object, the method behavior is modified as described in 4.2.2.2.1.1.

When a `Set` method targets the PendingAccessConditions column of a `K_AES_256` table object, the method behavior is modified as described in 4.2.2.2.2.

4.1.2.5 GenKey

This section describes changes to the `GenKey` method defined in [2]. As defined in section 5.3.3.16 in [2], this method causes an existing object in `C_AES_*`, `K_AES_256`, `C_EC_*`, `C_PIN`, `C_RSA_*`, or `C_HMAC_*` tables to be filled in with new key material.

This feature set extends the valid target objects to include some `Certificates` Table objects as defined in 4.2.2.1. Upon successful completion of the method, the target certificates object is filled with new key material.

This feature set defines a new AdditionalAccessConditions parameter that is a list of access_conditions_object_uidref values. This parameter SHALL only be defined when the method is invoked on a `K_AES_256` object.

After successful invocation of GenKey on a K_AES_256 object, the ActiveAccessConditions column of the object is updated as described in the AdditionalAccessConditions parameter description (see 4.1.2.5.1.1).

Any access to the media encryption key of the target K_AES_256 table object becomes constrained by the state of the AccessCondition objects in the ActiveAccessConditions list of the K_AES_256 table object as described in this feature set (see 4.2.2.2.1).

```
CredentialObjectUID.GenKey [
      PublicExponent = uinteger,
       PinLength = uinteger,
       AdditionalAccessConditions = list [ access_conditions_object_uidref …  ] ]
=>
[ ]
```

The parameter number for AdditionalAccessConditions SHALL be 0x070000.

Start of Informative Comment

The GenKey method does not cause any change to the underlying cryptographic material associated with any AccessCondition objects in the AdditionalAccessConditions list. The lifecycle of such material is controlled by the AccessCondition management commands described in section 4.1, as well as Locking SP lifecycle commands such as Revert and RevertSP.

End of Informative Comment

If the list length of the set union of the PendingAccessConditions column of the target K_AES_256 table object and AdditionalAccessConditions parameter is greater than Maximum Number of AccessConditions Per Key value reported in the feature set descriptor, then the method SHALL fail with a status of INVALID_PARAMETER.

Start of Informative Comment

This feature set requires that media encryption keys protected by AccessCondition objects always be protected cryptographically by the associated access_key(s). Allowing a locking disabled state transition where the ReadLockEnabled and WriteLockEnabled are set to False would require the TPer to have copies of the media encryption keys accessible without associated access_key(s), violating that property. So, a locking enabled state needs to be instantiated first before one can associate an AccessCondition object with a K_AES_256 table object via the GenKey method.

End of Informative Comment

If:

a) the union of PendingAccessConditions column of the target K_AES_256 table object and the AdditionalAccessConditions parameter is a non-empty list; and
b) the target K_AES_256 object is in the ActiveKey column of a Locking object that has a False value in the ReadLockEnabled or WriteLockEnabled columns;

then the method SHALL fail and return status of FAIL.

Start of Informative Comment

This feature set requires that media encryption keys protected by AccessCondition objects always be protected cryptographically by the associated encrypted_access_key(s). Any new media encryption keys generated can only have that property if any applicable AccessCondition object(s) have been enabled by providing the

encrypted_access_keys(s). TCG Opal permits successful host invocations of GenKey on locked Locking table objects and this property needs to be maintained (see [3]). However, host should be aware that a GenKey invoked on a locked Locking table object can only eradicate previous media encryption keys of that Locking table object and will not generate any new media encryption until the associated Locking object has been unlocked by the host.

End of Informative Comment

If:

a) any AccessCondition object in the PendingAccessConditions column has a False value in the AccessEnabled column; or
b) any AccessCondition object in the AdditionalAccessConditions parameter has a False value in the AccessEnabled column;

then:

a) the method SHALL complete successfully, eradicate all previous media encryption keys; and
b) the method SHALL NOT generate any new media encryption keys.

The TPer SHALL instead generate media encryption keys bounded by the AccessCondition objects specified in the AdditionalAccessConditions parameter and PendingAccessConditions when all of their AccessEnabled column values transition to True (see section 4.1.1.4) and the associated Locking object is in an unlocked state.

4.1.2.5.1 Parameter Descriptions

4.1.2.5.1.1 AdditionalAccessConditions

This parameter is only allowed when GenKey is called on a K_AES_256 table object. If the method is invoked on any other object types, then the method SHALL fail with a status of INVALID_PARAMETER.

When the ActiveAccessConditions column of the K_AES_256 table object is populated following a successful invocation of the GenKey method, it SHALL be set to the union of the PendingAccessConditions column and, if provided, the AdditionalAccessConditions parameter.

Any AccessCondition object references duplicated between the PendingAccessConditions column and the parameter SHALL appear exactly once in the resulting ActiveAccessConditions column value.

If the resulting list length is greater than the Maximum Number of AccessConditions Per Key value reported in the feature set descriptor (see section 5.2), the method SHALL fail with a status of INVALID_PARAMETER.

If the AdditionalAccessConditions parameter is not specified, or it is specified with an empty list, then the ActiveAccessConditions column of the K_AES_256 table object SHALL be populated by the AccessCondition object specified by the PendingAccessConditions column following a successful invocation of the GenKey method.

The PendingAccessConditions column value SHALL be retained across the method invocation.

4.2 Tables

This section defines new tables and modifications to existing tables required for this feature set.

4.2.1 New Tables

This feature set modifies the following tables:

a) AccessCondition

4.2.1.1 AccessCondition Table

Table 2: LockingSP - AccessCondition Table

Column Number	Column Name	IsUnique Column	Type
0x0	UID	(NR)[1]	uid
0x1	Name	Yes	name
0x2	CommonName	Yes	name
0x3	AccessEnabled	(NR)[1]	boolean
0x4	DisableAccessOnReset	(NR)[1]	reset_types
0x5	EncapsulationKey	(NR)[1]	kem_certificates_object_uidref

Notes:
1. The content of the cell is defined by the TCG Core Specification [2].

4.2.1.1.1 UID
This is the unique identifier for this `AccessCondition` object.

This column SHALL NOT be modifiable by the host.

4.2.1.1.2 Name
This is the name of the `AccessCondition` object.

4.2.1.1.3 CommonName
This is a name that MAY be shared by multiple `AccessCondition` objects.

4.2.1.1.4 AccessEnabled
The value of this column identifies the current access control state for this `AccessCondition` object.

The TPer SHALL modify the behavior of some method invocations as described in sections 4.2.2.2.1 and 4.2.2.2.2, based on the value of this column.

4.2.1.1.5 DisableAccessOnReset
This value defines the behavior of this AccessCondition object when reset events occur. The values enumerated in this column identify the reset types that cause the value of the AccessEnabled column of the AccessCondition object to be set to False.

The TPer SHALL change its locking state as described in the `DisableAccess` method description (see 4.1.1.5) if the AccessEnabled column value transitions from True to False as a result of one of the reset events in this column.

4.2.1.1.6 EncapsulationKey
The value of this column is a certificates_object_uidref value that corresponds to a KEM public key certificate in the `Certificates` table. A host is expected to use the KEM algorithm and public key reported in the certificate to produce the KEM_encapsulated_key and KEM_encapsulated_data values as described in the type description.

The value of this column MAY be unique to each AccessCondition object.

Start of Informative Comment

Whether a TPer implements different encapsulation keys for different AccessCondition objects depends on the level of isolation they wish to offer in multi-user use cases. A common KEM keypair used by all AccessCondition is likely to

be safe for many use cases, but in cases where the certificates objects have their underlying key rotated frequently in response to user management actions, this can result in failed method calls by different users if the keypair changes in between the user loading the X.509 certificate from the TPer and invoking a method call using derived KEM_encapsulated_data values.

End of Informative Comment

4.2.2 Modified Tables

This feature set modifies the following tables:

a) `Certificates`;

b) `K_AES_256`; and

c) `Locking`.

4.2.2.1 Certificates

A TPer implementing this feature set SHALL support the `Certificates` table from the base template as part of the Locking SP. This feature set modifies the `Certificates` table from the base template by adding the following columns (see Table 3), in addition to those defined in [2]:

Table 3: LockingSP - Certificates Table Columns

Column Number	Column Name	IsUnique	Column Type
0x05	EphemeralKeyPair	(NR)[1]	boolean
0x06	NextCertificate	(NR)[1]	Certificates_object_ref
Notes: 1. The content of the cell is defined by the TCG Core Specification [2].			

4.2.2.1.1 EphemeralKeyPair

The value of this column indicates whether the public key in the certificate data is part of an ephemeral key pair.

The TPer SHALL generate a new keypair and certificate for any object with a True value in this column in response to the `GenKey` method successfully invoked on the object.

The TPer SHALL eradicate the keypair and certificate for any object with a True value in this column in response to the following events:

- at each power cycle;
- as part of processing the Activate method for the LockingSP; and
- as part of processing the Reactivate method for the LockingSP.

4.2.2.1.2 NextCertificate

The value of this column identifies the next certificate in a certificate chain exported by the TPer.

If a TPer provides a certificate that is attested by another key provided by the TPer, this column MUST be populated with the UID of the corresponding `Certificates` object.

If the TPer provides a certificate that is not attested by another key provided by the TPer, the column value SHALL be a NULL UID (see TCG Core Spec section 3.2.5.3).

The TPer SHALL set the column value to a NULL UID to indicate a certificate chain termination.

4.2.2.1.3 KEM Certificates

KEM public key certificates indicated by the EncapsulationKey column values in the `AccessCondition` table SHALL set the EphemeralKeyPair column to True.

For KEM certificates, the encoding of the certificate data in the byte table SHALL be an X.509v3 DER certificate as defined by [4]. The X.509 public key certificate SHALL be attested by a cryptographic identity certificate bound to the Storage Device and SHALL NOT be self-signed (i.e., validate to the Storage Device vendor's Root Trusted Certificate Authority). The NextCertificate column SHALL be implemented for KEM public key certificates and SHALL indicate the cryptographic identity certificate bound to the Storage Device.

In addition, the X.509 extended attributes SHALL:

- indicate the key usage as keyEncipherment;
- include the HPKEIdentifiers structure, as defined in 4.2.2.1.3.1.

Start of Informative Comment

The number of KEM public key `Certificates` objects implemented to support this feature set will vary depending on target use cases of the TPer. In a minimal implementation of this feature set, the `Certificates` table will contain exactly one object with the EphemeralKeyPair column set to True, and that same object will be the EncapsulationKey column value for each object in the `AccessCondition` table. In an implementation with maximal isolation, each object of the `AccessCondition` table will have a unique object in the `Certificates` table assigned to it in the EncapsulationKey column, such that it is possible for each user given management of an AccessCondition object to have a unique KEM keypair to securely provision encrypted access_key values.

The cryptographic identity certificate used to attest the KEM public key certificate is a Vendor Unique value and can have multiple valid implementations. A self-signed certificate is invalid because it provides no receiver identity verification. A simple form of attestation would be a separate device certificate provisioned during manufacturing and attested by a public Certificate Authority.

A more robust implementation would implement the Implicit Identity Based Device Attestation specification (see [7]) and use the Alias Key to attest the KEM public key certificate. When using the Alias Key, it should be reported in the `Certificates` table so that it can be reported in any KEM certificates extended attributes.

End of Informative Comment

4.2.2.1.3.1 HPKEIdentifiers

As discussed in the previous section, the TPer SHALL include the HPKEIdentifiers structure as an extension to a KEM public key certificate exported by the TPer and reported in the `Certificates` table (see section 4.2.2.1).

The ASN.1 structure for the HPKEIdentifiers is defined as:

```
HPKEIdentifiers ::= SEQUENCE {
  kem_id INTEGER (0..65535), -- HPKE KEM ID (2 bytes, unsigned)
  kdf_id INTEGER   (0..65535), -- HPKE KDF ID (2 bytes, unsigned)
  aead_id INTEGER(0..65535)  -- HPKE AEAD ID (2 bytes, unsigned)
}
```

This extension uses an Object Identifiers (OIDs) arc from the TCG namespace.

```
TCG Branch:        tcg OBJECT IDENTIFIER ::= {2 23 tcg (133) }
Storage Branch:    tcg-storage OBJECT IDENTIFIER ::= { tcg storage (21) }
                   tcg-storage-hpke OBJECT IDENTIFIER ::= { tcg-storage hpke(1) }
```

The following OID SHALL be used to identify the HPKEIdentifiers structure:

```
tcg-storage-hpke-access-key-identifiers OBJECT IDENTIFIER ::= { tcg-storage-hpke
access-key-identifiers (1) }
```

The values for kem_id, kdf_id, and aead_id represent identifiers for Hybrid Public Key Encryption (HPKE), an algorithmic process which defines how to encrypt data to a KEM public key as defined by the ContextS.Seal() operation in section 5.2 of [6].

The extension is defined as:

Extension ::= SEQUENCE {

  extnID OBJECT IDENTIFIER, -- contains the DER encoding of the HPKEIdentifiers OID

  critical BOOLEAN DEFAULT FALSE,

  extnValue OCTET STRING -- contains the DER encoding of the HPKEIdentifiers structure

}

4.2.2.1.3.2 Additional considerations for the HPKE KEM Certificate

4.2.2.1.3.2.1 Serial Number

The Serial Number field SHALL be device unique.

4.2.2.1.3.2.2 Certificate Lifetime

As defined in the previous section, the KEM public key certificate is ephemeral, and its lifecycle is controlled by the host-driven events specified in section 4.2.2.1.1. As a result, the TPer SHALL set the X.509-defined GeneralizedTime value of 99991231235959Z in the notAfter portion of the certificate's Validity Period to indicate that the KEM public key certificate has no defined expiration date. The TPer can set the notBefore portion to a fixed known date and time in the past (e.g., Storage Device FW build time).

4.2.2.2 K_AES_256

This feature set modifies the `K_AES_256` table by adding the following columns (see Table 4), in addition to those defined in [2]:

Table 4: LockingSP - K_AES_256 Table Columns

Column Number	Column Name	IsUnique	Column Type
0x05	ActiveAccessConditions	(NR)[1]	list [access_condition_object_uidref …]
0x06	PendingAccessConditions	(NR)[1]	list [access_condition_object_uidref …]
Notes: 1. The content of the cell is defined by the TCG Core Specification [2].			

4.2.2.2.1 ActiveAccessConditions

This is a list of uidrefs for objects in the `AccessCondition` table. The state of the AccessCondition objects modifies the behavior of some methods as defined in the subsections below.

The value of this column SHALL be an empty list following a successful invocation of the `Revert` or `RevertSP` methods.

Start of Informative Comment

When an AccessCondition object in the ActiveAccessConditions list has an AccessEnabled column value of False, methods that require use of any media encryption key managed by the `K_AES_256` table object will not be allowed (see 4.2.2.2.1.1).

Additionally, for any `Locking` table object where the ActiveKey is set to this `K_AES_256` table object, the result of any `Set` method that changes the ReadLockEnabled or WriteLockEnabled value from True to False is constrained as described in 4.2.2.2.1.1.

End of Informative Comment

4.2.2.2.1.1 Set Method

Start of Informative Comment

This section describes how AccessCondition objects in the ActiveAccessConditions column constrain changing the ReadLockEnabled and WriteLockEnabled columns of a Locking object. This feature set requires that media encryption keys protected by AccessCondition objects always be protected cryptographically by the associated access_key(s). Disabling locking would require the TPer to have copies of the media encryption keys accessible without requiring the access_key(s), violating that property.

End of Informative Comment

If:

- an AccessCondition object is present in the ActiveAccessConditions column of a `K_AES_256` table object;
- the `K_AES_256` table object is indicated by the ActiveKey column of a Locking object;
- the `Set` method is invoked on the Locking object;
- the `Set` method parameters specify either
  - the ReadLockedEnabled column; or
  - the WriteLockedEnabled column;
- the currently configured column value is True; and
- the specified column value is False;

Then the Set method SHALL fail with a status of FAIL.

Start of Informative Comment

This clause describes how an AccessCondition object constrains changing the ReadLocked and WriteLocked columns of a Locking object.

End of Informative Comment

If:

- an AccessCondition object is present in the ActiveAccessConditions column of a `K_AES_256` table object;
- the `K_AES_256` table object is indicated by the ActiveKey column of a Locking object;
- the `Set` method is invoked on the Locking object;
- the `Set` method parameters specify either
  - the ReadLocked column; or
  - the WriteLocked column;
- the currently configured column value is True;

- the specified column value is False; and
- the AccessEnabled value for the AccessCondition object is False

Then, the `Set` method SHALL fail with a status of NOT_AUTHORIZED.

4.2.2.2.2 PendingAccessConditions

Start of Informative Comment

This section describes how Maximum Number of AccessConditions Per Key value reported in the feature set descriptor constrains `Set` method invoked on this column and when this column values are zeroized.

End of Informative Comment

This is a list of uidrefs for objects in the `AccessCondition` table. The state of the PendingAccessConditions objects modifies the behavior of some methods as defined in the subsections below.

If:

- the `Set` method is invoked to change the value of this column; and
- the target list length is greater than the Maximum Number of AccessConditions Per Key value reported in the feature set descriptor.

Then the method SHALL fail with a status of INVALID_PARAMETER.

The value of this column SHALL be an empty list following a successful invocation of the `Revert` or `RevertSP` methods.

4.2.2.2.2.1 GenKey Method

Start of Informative Comment

This section describes how AccessCondition object states constrain the generation of new Media Encryption Keys using the `GenKey` method. All AccessCondition objects in the PendingAccessConditions column and AdditionalAccessConditions parameter (see 4.1.2.5.1.1) need to be in an enabled state to generate new media encryption keys. Otherwise, `GenKey` method will only eradicate previous media encryption keys but will not generate any new media encryption keys until the associated Locking object is in an unlocked state.

End of Informative Comment

If:

a) an AccessCondition object is present in the PendingAccessConditions column of a `K_AES_256` table object;
b) the AccessEnabled column value for the AccessCondition object is False; and
c) the `GenKey` method is invoked on the `K_AES_256` table object

Then:

a) the `GenKey` method SHALL complete successfully;
b) the method SHALL eradicate all previous media encryption keys; and
c) the method SHALL NOT generate any new media encryption keys.

The TPer SHALL instead generate media encryption keys bounded by the AccessCondition objects specified in the AdditionalAccessConditions parameter and PendingAccessConditions when all of their AccessEnabled column values transition to True (see section 4.1.1.4) and the associated Locking object is in an unlocked state.

4.2.2.3 Locking

As defined in [3], all `Locking` table objects have an associated `K_AES_256` table object as defined by the ActiveKey column of the `Locking` table object.

When any AccessCondition object in the ActiveAccessConditions column of that `K_AES_256` table object transitions from an AccessEnabled value of True to an AccessEnabled value of False, it SHALL cause the TPer to transition the ReadLocked and WriteLocked columns of that `Locking` table object to True.

Locking objects where the `K_AES_256` table object in the ActiveKey column contains a non-empty ActiveAccessConditions column SHALL have their WriteLockEnabled and ReadLockEnabled columns set to True.

Start of Informative Comment

The `GenKey` method in this feature set also requires WriteLockEnabled and ReadLockEnabled columns to be True to successfully execute the method when it results in a non-empty ActiveAccessConditions list. Similarly, the `Set` method in this feature set also states that neither the WriteLockEnabled nor ReadLockEnabled columns can be set to False while the ActiveAccessConditions list is non-empty.

This feature set defines multiple ways for the AccessEnabled state to transition from True to False, including explicitly through the `DisableAccess` method and implicitly through a reset event that is present in the DisableAccessOnReset column. Any such transition will result in the media encryption keys of `K_AES_256` table objects that have the AccessCondition object in their ActiveAccessConditions column becoming inaccessible, as defined in the `K_AES_256` table modifications section of this feature set.

End of Informative Comment

4.3 Types

This section defines new types and modifications to existing types required for this feature set.

4.3.1 New Types

4.3.1.1 Abstract Types

4.3.1.1.1 KEM_encapsulated_key

This abstract data type is the encapsulated encryption key output of a Key Encapsulation Mechanism (KEM) Encap function and corresponds to the enc output of the SetupBaseS() operation described in section 5.2 of [6]. The length of this data type is dependent on the kem_id value encoded in the HPKEIdentifiers (see section 4.2.2.1.3.1).

Format:

```
Bytes
```

Start of Informative Comment

[6] gives examples of the expected length in bytes of an encapsulated key produced by a given KEM algorithm. For example, if DHKEM(P-521, HKDF-SHA512) is KEM algorithm supported by the TPer, then the length of encapsulated key value specified by the host would be N = 133 bytes. Different kem_id values (e.g., (DHKEM(P-384, HKDF-SHA384, DHKEM(P-256, HKDF-SHA256), etc..) would provide a different N value.

Note that the size of the KEM encapsulated key can potentially be rather large for modern post-quantum algorithms, especially when used as part of hybrid algorithms, and that size may potentially require changes to TPer's supported token and ComPacket size limits (see section 5.3).

End of Informative Comment

4.3.1.1.2 KEM_encapsulated_data

This abstract data type indicates the raw output (encrypted data, plus tag when applicable) of applying the AEAD encryption algorithm, indicated by the aead_id value encoded in the HPKEIdentifiers extension of the KEM public key certificate indicated by the EncapsulationKey column of the target `AccessCondition` object, to the data being encapsulated.

This corresponds to the ct value returned from ContextS.Seal() operation described in section 5.2 of [6]. When used to derive key material, the mode value as specified is mode_base, as defined in table 1 of section 5 in [6].

The length this data type is dependent on the aead_id value encoded in the HPKEIdentifiers (see section 4.2.2.1.3.1) and the length of the plain text value that gets encrypted by the aead_id algorithm.

The abstract type takes in a sequence_number and the access_key value. The sequence_number is used to indicate the sequence of the encryption operation being performed using the same HPKE context.

As section 5.2 of [6] describes, when using the same context to perform multiple encryptions (e.g., see section 4.1.1.3), the sequence_number, which starts at 0 by default, is expected to be incremented by 1 for each message encrypted with the context to provide ciphertext uniqueness. This property allows the creator of the first message to prove that they also created the next message.

Format:

`Bytes`

Start of Informative Comment

Most aead algorithms typically produce a ciphertext length that equals to the original plaintext length plus an authentication tag. The latter may be 16 bytes long (e.g., GCM mode) or 8 or 12 bytes long (e.g., CCM mode). Because of this, the expected length of KEM_encapsulated_data will vary based on the aead_id value encoded in the HPKEIdentifiers extension of the KEM public key certificate indicated by the EncapsulationKey column of the target AccessCondition object.

End of Informative Comment

The AEAD encryption key is generated by applying the key derivation function (KDF) indicated by the kdf_id value encoded in the HPKEIdentifiers to the Shared Secret output of a Key Encapsulation Mechanism (KEM) Decap function to the KEM_encapsulated_key passed with the KEM_encapsulated_data. The KEM algorithm used is defined by the kem_id value encoded in the HPKEIdentifiers.

The TPer SHALL use the ContextR.Open() operation described in section 5.2 of [6] to get the data encapsulated in the KEM_encapsulated_data.

The TPer SHALL decapsulate the payload prior to returning status for the method call which included the encapsulated data, such that no dependency on the KEM keypair used to encapsulate the data exists after the method call completes.

Start of Informative Comment

A host needs to retrieve the following 3 pieces of information from the TPer to successfully generate a value of the KEM_encapsulated_data type:

a) A KEM encryption algorithm and the associated public key.

   a. The TPer reports HPKE public key and a KEM algorithm identifier (i.e., the kem_id value) in the HPKEIdentifiers extension of the KEM public key certificate indicated by the EncapsulationKey column of the target AccessCondition object.

b) A Key Derivation Function (KDF) to be used with the KEM to transform the KEM Shared Secret into a Key Encryption key.

   a. The TPer reports the key_derivation_algorithm_identifier value for the algorithm it supports (i.e., kdf_id value) in the HPKEIdentifiers extension of the KEM public key certificate indicated by the EncapsulationKey column of the target AccessCondition object.

c) An AEAD Encryption algorithm to be used to encrypt the encapsulated data with the derived encryption key produced by the KDF.

   a. The TPer reports the aead encryption algorithm identifier value for the algorithm it supports (i.e., aead_id value) in the HPKEIdentifiers extension of the KEM public key certificate indicated by the EncapsulationKey column of the target AccessCondition object.

For the TPer to decrypt encapsulated data, the KEM_encapsulated_key value is expected to be passed as a parameter along with any KEM_encapsulated_data values.

The requirement to decapsulate the payload during the method invocation exists to ensure that a subsequent `GenKey` method call on the certificates object for the KEM keypair does not make the payload inaccessible. When the payload needs to persist for later use by the TPer, it is expected to be stored in a key vault or re-encrypted with an ephemeral key stored in a key vault.

End of Informative Comment

4.3.1.2 Restricted Type

4.3.1.2.1 access_condition_object_uidref

The access_condition_object_uidref type is a restricted reference type that SHALL be used specifically for uidrefs to AccessCondition objects.

Table 5: access_condition_object_uidref

UID	Name	Format
00 00 00 05 00 00 21 01	access_conditions_object_uidref	Restricted_Reference_Type{6}, uidref {AccessConditionTableUID}

4.3.1.2.2 kem_certificates_object_uidref

The kem_certificates_object_uidref type is a restricted reference type that SHALL be used specifically for uidrefs to `Certificates` objects.

Table 6: kem_certificates_object_uidref

UID	Name	Format
00 00 00 05 00 00 21 02	kem_certificates_object_uidref	Restricted_Reference_Type{6}, uidref {CertificatesTableUID}

4.3.2 Modified Types

There are no types modified by this feature set.

5 Feature Set Requirements

This section defines the requirements for the MEK Multiparty Authorization Feature Set.

5.1 Requirements Overview

The MEK Multiparty Authorization Feature Set consists of multiparty authorization capabilities that MAY be implemented in a TPer. A host discovers the MEK Multiparty Authorization specific capabilities and properties of a TPer by examining the Level 0 discovery feature set descriptor.

5.2 Level 0 Discovery

A SD implementing the MEK Multiparty Authorization Feature Set SHALL:

a) return the MEK Multiparty Authorization Feature Set descriptor in the Level 0 discovery response.

5.2.1 MEK Multiparty Authorization Feature Descriptor (Feature Code = 0x040C)

The contents of the feature descriptor are defined in this section below.

Table 7: Level0 Discovery - MEK Multiparty Authorization Feature Descriptor

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)			Feature Code				
1								(LSB)
2	Feature Descriptor Version Number				Feature Set Minor Version Number			
3	Length							
4	(MSB)			Number of Access Conditions				
5								(LSB)
6	(MSB)			Maximum Number of Access Conditions Per Key				
7								(LSB)
8	Access Key Length							
9	Reserved							HW Reset for DisableAccessOnReset Supported
10				Reserved				
11								

5.2.1.1 Feature Code

The Feature Code field value SHALL be set to `0x040C`.

5.2.1.2 Feature Descriptor Version Number

The Feature Descriptor Version Number field SHALL be set to `0x1`.

5.2.1.3 Feature Set Minor Version Number

The Feature Set Minor Version Number represents the minor version of the MEK Multiparty Authorization Feature Set supported by the SD.

The Feature Set Minor Version Number field SHALL be set to a value as specified in Table 8

Table 8: Feature Set Minor Versions

Feature Set Minor Version Number	Specification Referenced
0x0	MEK Multiparty Authorization Feature Set v1.00
All others	Reserved

5.2.1.4 Length

The Length field indicates the number of bytes in the descriptor following byte 3. The value of the Length field SHALL be 8.

5.2.1.5 Number of AccessConditions

The Number of AccessConditions field indicates the total number of AccessCondition objects that a TPer supports. This value SHALL be set to the number of rows present in the `AccessCondition` table.

The value of this field SHALL be set to 1 or greater.

5.2.1.6 Maximum Number of AccessConditions Per Key

The Maximum Number of AccessConditions Per Key field indicates the maximum number of AccessCondition objects that can be configured per `K_AES_256` table object (i.e., the maximum list length of the ActiveAccessConditions column values and PendingAccessConditions column values in the `K_AES_256` table row).

The value of this field SHALL be set to 1 or greater.

5.2.1.7 Access Key Length

This is the Vendor Unique length in bytes of the encrypted access_key values that are encapsulated and passed as a parameter for the `InitializeAccessCondition`, `EnableAccess`, `TestAccessKey`, and `ChangeAccessKey` methods.

The value of this field SHALL be set to 32 or greater.

5.2.1.8 HW Reset for DisableAccessOnReset Supported

The HW Reset for DisableAccessOnReset Supported field indicates whether the TPer supports the Hardware Reset value for the `AccessCondition` table's DisableAccessOnReset column (see section 5.5.1.1.6.1 for details).

The value of this field SHALL be set to one if the TPer supports the Hardware Reset value for the DisableAccessOnReset column. Otherwise, the value of this field SHALL be set to zero.

5.3 Communication Properties

Start of Informative Comment

As discussed in previous sections, this feature set relies on HPKE (see [6]) to protect host provisioned secrets as they are sent across the TPer's interface.

Thus, TPer implementations that are able to account for the size of both the KEM_encapsulated_data types and the KEM_encapsulated_key types imposed by HPKE KEM algorithms selected can increase the values of relevant TPer Properties accordingly (see section 4.1 in [2] for `Properties` method and Property Names).

The total number of bytes required for the various types depends on the selected algorithms, their associated key lengths, and other factors. Therefore, no specific value is specified as required by this specification, as it may not be sufficient for all implementations.

There are three Property values to which attention is called in particular:

Table 9: Communication Properties Considerations

Property	Minimum Value	Applicable To
MaxIndTokenSize	The minimum value specified in [3] may need to be increased to allow passing large KEM_encapsulated_key values as a single token	Host and TPer Property
MaxPacketSize	The minimum values specified in [3] may need to be increased to accommodate methods that include multiple KEM_encapsulated_key values and KEM_encapsulated_data values specified in section 4.1.	
MaxComPacketSize		

End of Informative Comment

5.4 Admin SP

This feature set modifies the behavior of the `Activate` method (see 4.1.2.1), `Revert` method (see 4.1.2.2), and `RevertSP` method (see 4.1.2.3).

5.5 Locking SP

A SD implementing this feature set SHALL support the additions to the Locking SP specified in this section, in addition to the Locking SP requirements defined in [3].

5.5.1 Tables Preconfiguration

5.5.1.1.1 MethodID

The MethodID Table is defined in [2]. In addition to the requirements in [3], the `MethodID` Table Preconfiguration data SHALL be modified as specified in Table 10.

Table 10: LockingSP - MethodID Table Preconfiguration

UID[RF]	Name[1]	CommonName	TemplateID
00 00 00 06 00 00 0B 01	“InitializeAccessCondition”	See note 2	See note 2
00 00 00 06 00 00 0B 02	“TestAccessKey”	See note 2	See note 2
00 00 00 06 00 00 0B 03	“ChangeAccessKey”	See note 2	See note 2
00 00 00 06 00 00 0B 04	“EnableAccess”	See note 2	See note 2
00 00 00 06 00 00 0B 05	“DisableAccess”	See note 2	See note 2

Notes:
1. The text in each table cell in the column is informative. See the TCG Core Specification [2] the applicable TCG Opal Family SSC.
2. The content of the cell is defined by the TCG Core Specification [2] and the applicable TCG Opal Family SSC.
3. RF: The content of each cell in the column is Read-only as specified in the applicable TCG Opal Family SSC.

5.5.1.1.2 Table

The Table Table [is defined in [2]. In addition to the requirements in [3], the `Table` Table Preconfiguration data SHALL be modified as specified in Table 11.

Table 11: LockingSP - Table Table Preconfiguration

UID[RF]	Name[RF]	CommonName	TemplateID	Kind[RF]	Column	NumColumns	Rows	RowsFree	RowBytes	LastId	MinSize	MaxSize	MandatoryWrite	RecommendedAccess
00 00 00 01 00 00 17 01	"AccessCondition"	See note 2	(NR)[3]	Object	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	0[RF]	0[RF]
00 00 00 01 00 00 00 0A	"Certificates"	See note 2	(NR)[3]	Object	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	0[RF]	0[RF]
00 00 00 01 00 00 18 01	"CertificateData1"	See note 2	(NR)[3]	Byte	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	VU[RFV]	VU[RFV]
00 00 00 01 00 00 18 01 (+ YYYY)	"CertificateDataYYYY"[1]	See note 2	(NR)[3]	Byte	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	(NR)[3]	VU[RFV]	VU[RFV]

Notes:
1. CertificateDataYYYY objects are added to the `Table` Table where YYYY can range from 2 to 1024.
2. The content of the cell is defined by the TCG Core Specification [2] and the applicable TCG Opal Family SSC.
3. The content of the cell is not required. See the TCG Core Specification [2] and the applicable Opal Family SSC.
4. RF: For table cells, the content of the cell is Read-only as specified by the applicable TCG Opal Family SSC. For table columns, the content of each cell in the column is Read-only as specified by the applicable TCG Opal Family SSC.
5. RFV: The content of the cell is Read-only as specified by the applicable TCG Opal Family SSC. The value of the cell is vendor unique.

5.5.1.1.3 AccessControl

The AccessControl Table is defined in [2]. In addition to the requirements in [3], the `AccessControl` Table Preconfiguration data SHALL be modified as specified in Table 12:

Table 12: LockingSP - AccessControl Table Preconfiguration

Table Association - informative only	UID	InvokingID[RF]	InvokingID Name - informative only[1]	MethodID[RF]	CommonName	ACL[RF]	Log	AddACEACL	RemoveACEACL	GetACLACL[RF]	DeleteMethodACL	AddACELog	RemoveACELog	GetACLLog	DeleteMethodLog	LogTo
ACE																
	(ND)[3]	00 00 00 08 00 06 40 02	ACE_K_AES_256_GlobalRange_Set_PendingAccessConditions	Set	See note 2	ACE_ACE_Set_BooleanExpression	(NR)[4]	(NR)[4]	(NR)[4]	ACE_Anybody	(NR)[4]	(NR)[4]	(NR)[4]	(NR)[4]	(NR)[4]	(NR)[4]
	(ND)[3]	00 00 00 08 00 06 41 01	ACE_K_AES_256_Range1_Set_PendingAccessConditions	Set	See note 2	ACE_ACE_Set_BooleanExpression	(NR)[4]	(NR)[4]	(NR)[4]	ACE_Anybody	(NR)[4]	(NR)[4]	(NR)[4]	(NR)[4]	(NR)[4]	(NR)[4]

		Table Association - informative only
(ND)[3]	(ND)[3]	**UID**
00 00 00 08 00 06 42 01	00 00 00 08 00 06 41 00 (+NNNN)	**InvokingID[RF]**
ACE_AccessCondition_Conditions1_Set_DisableOnReset	ACE_K_AES_256_RangeNNNN_Set_PendingAccessConditions	**InvokingID Name - informative only[1]**
Set	Set	**MethodID[RF]**
See note 2	See note 2	**CommonName**
ACE_ACE_Set_BooleanExpression	ACE_ACE_Set_BooleanExpression	**ACL[RF]**
(NR)[4]	(NR)[4]	**Log**
(NR)[4]	(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]	**LogTo**

		Table Association - informative only
(ND)[3]	(ND)[3]	**UID**
00 00 00 08 00 06 43 01	00 00 00 08 00 06 42 00 (+XXXX)	**InvokingID[RF]**
ACE_AccessCondition_Conditions1 _InitializeAccessCondition	ACE_AccessCondition_ConditionsX XXX_Set_DisableOnReset	**InvokingID Name - informative only[1]**
Set	Set	**MethodID[RF]**
See note 2	See note 2	**CommonName**
ACE_ACE_Set_BooleanExpression	ACE_ACE_Set_BooleanExpression	**ACL[RF]**
(NR)[4]	(NR)[4]	**Log**
(NR)[4]	(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]	**LogTo**

		Table Association - informative only
(ND)[3]	(ND)[3]	**UID**
00 00 00 08 00 06 44 01	00 00 00 08 00 06 43 00 (+XXXX)	**InvokingID[RF]**
ACE_AccessCondition_Conditions1 _TestccessKey	ACE_AccessCondition_Conditions XXXX _InitializeAccessCondition	**InvokingID Name - informative only[1]**
Set	Set	**MethodID[RF]**
See note 2	See note 2	**CommonName**
ACE_ACE_Set_BooleanExpression	ACE_ACE_Set_BooleanExpression	**ACL[RF]**
(NR)[4]	(NR)[4]	**Log**
(NR)[4]	(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]	**LogTo**

		Table Association - informative only
(ND)[3]	(ND)[3]	**UID**
00 00 00 08 00 06 45 01	00 00 00 08 00 06 44 00 (+XXXX)	**InvokingID[RF]**
ACE_AccessCondition_Conditions1 _ChangeAccessKey	ACE_AccessCondition_Conditions XXXX _TestccessKey	**InvokingID Name - informative only[1]**
Set	Set	**MethodID[RF]**
See note 2	See note 2	**CommonName**
ACE_ACE_Set_BooleanExpression	ACE_ACE_Set_BooleanExpression	**ACL[RF]**
(NR)[4]	(NR)[4]	**Log**
(NR)[4]	(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]	**LogTo**

		Table Association - informative only
(ND)[3]	(ND)[3]	**UID**
00 00 00 08 00 06 46 01	00 00 00 08 00 06 45 00 (+XXXX)	**InvokingID[RF]**
ACE_AccessCondition_Conditions1 _EnableAccess	ACE_AccessCondition_Conditions XXXX _ChangeAccessKey	**InvokingID Name - informative only[1]**
Set	Set	**MethodID[RF]**
See note 2	See note 2	**CommonName**
ACE_ACE_Set_BooleanExpression	ACE_ACE_Set_BooleanExpression	**ACL[RF]**
(NR)[4]	(NR)[4]	**Log**
(NR)[4]	(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]	**LogTo**

		Table Association - informative only
(ND)[3]	(ND)[3]	**UID**
00 00 00 08 00 06 47 01	00 00 00 08 00 06 46 00 (+XXXX)	**InvokingID[RF]**
ACE_AccessCondition_Conditions1 _DisableAccess	ACE_AccessCondition_Conditions XXXX _EnableAccess	**InvokingID Name - informative only[1]**
Set	Set	**MethodID[RF]**
See note 2	See note 2	**CommonName**
ACE_ACE_Set_BooleanExpression	ACE_ACE_Set_BooleanExpression	**ACL[RF]**
(NR)[4]	(NR)[4]	**Log**
(NR)[4]	(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]	**LogTo**

	K_AES_256		**Table Association - informative only**
(ND)[3]		(ND)[3]	**UID**
00 00 08 06 00 00 00 01		00 00 00 08 00 06 47 00 (+XXXX)	**InvokingID[RF]**
K_AES_256_GlobalRange_Key		ACE_AccessCondition_Conditions XXXX _DisableAccess	**InvokingID Name - informative only[1]**
Get		Set	**MethodID[RF]**
See note 2		See note 2	**CommonName**
ACE_K_AES_AccessConditions		ACE_ACE_Set_BooleanExpression	**ACL[RF]**
(NR)[4]		(NR)[4]	**Log**
(NR)[4]		(NR)[4]	**AddACEACL**
(NR)[4]		(NR)[4]	**RemoveACEACL**
ACE_Anybody		ACE_Anybody	**GetACLACL[RF]**
(NR)[4]		(NR)[4]	**DeleteMethodACL**
(NR)[4]		(NR)[4]	**AddACELog**
(NR)[4]		(NR)[4]	**RemoveACELog**
(NR)[4]		(NR)[4]	**GetACLLog**
(NR)[4]		(NR)[4]	**DeleteMethodLog**
(NR)[4]		(NR)[4]	**LogTo**

		Table Association - informative only
(ND)[3]	(ND)[3]	**UID**
00 00 08 06 00 03 00 00 (+NN NN)	00 00 08 06 00 03 00 01	**InvokingID[RF]**
K_AES_256_RangeNNNN_Key	K_AES_256_Range1_Key	**InvokingID Name - informative only[1]**
Get	Get	**MethodID[RF]**
See note 2	See note 2	**CommonName**
ACE_K_AES_AccessConditions	ACE_K_AES_AccessConditions	**ACL[RF]**
(NR)[4]	(NR)[4]	**Log**
(NR)[4]	(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]	**LogTo**

		Table Association - informative only
(ND)[3]	(ND)[3]	**UID**
00 00 08 06 00 03 00 01	00 00 08 06 00 00 00 01	**InvokingID[RF]**
K_AES_256_Range1_Key	K_AES_256_GlobalRange_Key	**InvokingID Name - informative only[1]**
Set	Set	**MethodID[RF]**
See note 2	See note 2	**CommonName**
ACE_K_AES_256_Range1_Set_PendingAccessConditions	ACE_K_AES_256_GlobalRange_Set_PendingAccessConditions	**ACL[RF]**
(NR)[4]	(NR)[4]	**Log**
(NR)[4]	(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]	**LogTo**

	AccessCondition		**Table Association - informative only**
(ND)[3]		(ND)[3]	**UID**
00 00 17 01 00 00 00 00		00 00 08 06 00 03 00 00 (+NN NN)	**InvokingID[RF]**
AccessCondition		ACE_K_AES_256_RangeNNNN_Key	**InvokingID Name - informative only[1]**
Next		Set	**MethodID[RF]**
See note 2		See note 2	**CommonName**
ACE_Anybody		ACE_K_AES_256_RangeNNNN_Set_PendingAccessConditions	**ACL[RF]**
(NR)[4]		(NR)[4]	**Log**
(NR)[4]		(NR)[4]	**AddACEACL**
(NR)[4]		(NR)[4]	**RemoveACEACL**
ACE_Anybody		ACE_Anybody	**GetACLACL[RF]**
(NR)[4]		(NR)[4]	**DeleteMethodACL**
(NR)[4]		(NR)[4]	**AddACELog**
(NR)[4]		(NR)[4]	**RemoveACELog**
(NR)[4]		(NR)[4]	**GetACLLog**
(NR)[4]		(NR)[4]	**DeleteMethodLog**
(NR)[4]		(NR)[4]	**LogTo**

			Table Association - informative only
(ND)[3]	(ND)[3]	(ND)[3]	**UID**
00 00 17 01 00 00 00 01	00 00 17 01 00 00 00 00 (+XXXX)	00 00 17 01 00 00 00 01	**InvokingID[RF]**
AccessConditionObj	AccessConditionObj	AccessConditionObj	**InvokingID Name - informative only[1]**
Set	Get	Get	**MethodID[RF]**
See note 2	See note 2	See note 2	**CommonName**
ACE_AccessCondition_AccessCondition1_Set_DisableOnReset,ACE_Admins_Set_CommonName	ACE_Anybody	ACE_Anybody	**ACL[RF]**
(NR)[4]	(NR)[4]	(NR)[4]	**Log**
(NR)[4]	(NR)[4]	(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]	(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody	ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]	(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]	(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]	(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]	(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]	(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]	(NR)[4]	**LogTo**

		Table Association - informative only
(ND)[3]	(ND)[3]	**UID**
00 00 17 01 00 00 00 01	00 00 17 01 00 00 00 00 (+XXXX)	**InvokingID[RF]**
AccessConditionObj	AccessConditionObj	**InvokingID Name - informative only[1]**
InitializeAccessCondition	Set	**MethodID[RF]**
See note 2	See note 2	**CommonName**
ACE_AccessCondition_AccessCondition1_InitializeAccessCondition	ACE_AccessCondition_ConditionsXXXX_Set_DisableOnReset,ACE_Admins_Set_CommonName	**ACL[RF]**
(NR)[4]	(NR)[4]	**Log**
(NR)[4]	(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]	**LogTo**

		Table Association - informative only
(ND)[3]	(ND)[3]	**UID**
00 00 17 01 00 00 00 01	00 00 17 01 00 00 00 00 (+XXXX)	**InvokingID[RF]**
AccessConditionObj	AccessConditionObj	**InvokingID Name - informative only[1]**
TestAccessKey	InitializeAccessCondition	**MethodID[RF]**
See note 2	See note 2	**CommonName**
ACE_AccessCondition_AccessCondition1_TestAccessKey	ACE_AccessCondition_AccessConditionXXXX_InitializeAccessCondition	**ACL[RF]**
(NR)[4]	(NR)[4]	**Log**
(NR)[4]	(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]	**LogTo**

		Table Association - informative only
(ND)[3]	(ND)[3]	**UID**
00 00 17 01 00 00 00 01	00 00 17 01 00 00 00 00 (+XXXX)	**InvokingID[RF]**
AccessConditionObj	AccessConditionObj	**InvokingID Name - informative only[1]**
ChangeAccessKey	TestAccessKey	**MethodID[RF]**
See note 2	See note 2	**CommonName**
ACE_AccessCondition_AccessCondition1_ChangeAccessKey	ACE_AccessCondition_AccessConditionXXXX_TestAccessKey	**ACL[RF]**
(NR)[4]	(NR)[4]	**Log**
(NR)[4]	(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]	**LogTo**

		Table Association - informative only
(ND)[3]	(ND)[3]	**UID**
00 00 17 01 00 00 00 01	00 00 17 01 00 00 00 00 (+XXXX)	**InvokingID[RF]**
AccessConditionObj	AccessConditionObj	**InvokingID Name - informative only[1]**
EnableAccess	ChangeAccessKey	**MethodID[RF]**
See note 2	See note 2	**CommonName**
ACE_AccessCondition_AccessCondition1_EnableAccess	ACE_AccessCondition_AccessConditionXXXX_ChangeAccessKey	**ACL[RF]**
(NR)[4]	(NR)[4]	**Log**
(NR)[4]	(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]	**LogTo**

		Table Association - informative only
(ND)[3]	(ND)[3]	**UID**
00 00 17 01 00 00 00 01	00 00 17 01 00 00 00 00 (+XXXX)	**InvokingID[RF]**
AccessConditionObj	AccessConditionObj	**InvokingID Name - informative only[1]**
DisableAccess	EnableAccess	**MethodID[RF]**
See note 2	See note 2	**CommonName**
ACE_AccessCondition_AccessCondition1_DisableAccess	ACE_AccessCondition_AccessConditionXXXX_EnableAccess	**ACL[RF]**
(NR)[4]	(NR)[4]	**Log**
(NR)[4]	(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]	**LogTo**

			Certificates		**Table Association - informative only**
(ND)[3]	(ND)[3]	(ND)[3]		(ND)[3]	**UID**
00 00 00 0A 00 00 04 00 (+MMMM)	00 00 00 0A 00 00 04 01	00 00 00 0A 00 00 00 00		00 00 17 01 00 00 00 00 (+XXXX)	**InvokingID[RF]**
CertificateMMMM	Certificate1	Certificates		AccessConditionObj	**InvokingID Name - informative only[1]**
Get	Get	Next		DisableAccess	**MethodID[RF]**
See note 2	See note 2	See note 2		See note 2	**CommonName**
ACE_Anybody	ACE_Anybody	ACE_Anybody		ACE_AccessCondition_AccessConditionXXXX_DisableAccess	**ACL[RF]**
(NR)[4]	(NR)[4]	(NR)[4]		(NR)[4]	**Log**
(NR)[4]	(NR)[4]	(NR)[4]		(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]	(NR)[4]		(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody	ACE_Anybody		ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]	(NR)[4]		(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]	(NR)[4]		(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]	(NR)[4]		(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]	(NR)[4]		(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]	(NR)[4]		(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]	(NR)[4]		(NR)[4]	**LogTo**

		CertificateData			**Table Association - informative only**
(ND)[3]	(ND)[3]		(ND)[3]	(ND)[3]	**UID**
00 00 18 00 00 00 00 00 (+ YYYY * 2^{32})	00 00 18 01 00 00 00 00		00 00 00 0A 00 00 04 00 (+MMMM)	00 00 00 0A 00 00 04 01	**InvokingID[RF]**
CertificateDataYYYY	CertificateData1		CertificateMMMM	Certificate1	**InvokingID Name - informative only[1]**
Get	Get		Genkey	Genkey	**MethodID[RF]**
See note 2	See note 2		See note 2	See note 2	**CommonName**
ACE_Anybody	ACE_Anybody		ACE_Admin	ACE_Admin	**ACL[RF]**
(NR)[4]	(NR)[4]		(NR)[4]	(NR)[4]	**Log**
(NR)[4]	(NR)[4]		(NR)[4]	(NR)[4]	**AddACEACL**
(NR)[4]	(NR)[4]		(NR)[4]	(NR)[4]	**RemoveACEACL**
ACE_Anybody	ACE_Anybody		ACE_Anybody	ACE_Anybody	**GetACLACL[RF]**
(NR)[4]	(NR)[4]		(NR)[4]	(NR)[4]	**DeleteMethodACL**
(NR)[4]	(NR)[4]		(NR)[4]	(NR)[4]	**AddACELog**
(NR)[4]	(NR)[4]		(NR)[4]	(NR)[4]	**RemoveACELog**
(NR)[4]	(NR)[4]		(NR)[4]	(NR)[4]	**GetACLLog**
(NR)[4]	(NR)[4]		(NR)[4]	(NR)[4]	**DeleteMethodLog**
(NR)[4]	(NR)[4]		(NR)[4]	(NR)[4]	**LogTo**

Table Association - informative only	UID	InvokingID[RF]	InvokingID Name - informative only[1]	MethodID[RF]	CommonName	ACL[RF]	Log	AddACEACL	RemoveACEACL	GetACLACL[RF]	DeleteMethodACL	AddACELog	RemoveACELog	GetACLLog	DeleteMethodLog	LogTo

Notes:

1. The text in all table cells in the column is informative.
2. The content of the cell is defined by the TCG Core Specification [2] and the applicable TCG Opal Family SSC.
3. The content of the cell is not defined.
4. The content of the cell is not required. See the TCG Core Specification [2] and the applicable Opal Family SSC.
5. RF: The content of each cell in the column is Read-only as specified by the TCG Core Specification [2].

5.5.1.1.4 Access Control Element (ACE)

The ACE Table is defined in [2]. In addition to the requirements in [3], the `ACE` Table Preconfiguration data SHALL be modified as specified in Table 13.

Table 13: LockingSP - ACE Table Preconfiguration

Table Association -Informative Column	UID[RF]	Name[1]	CommonName	BooleanExpr	Columns[RF]
K_AES_256					
	00 00 00 08 00 06 40 01	"ACE_K_AES_AccessConditions"	See note 2	Anybody[RF]	ActiveAccessConditions,PendingAccessConditions

Table Association -Informative Column	UID[RF]	Name[1]	CommonName	BooleanExpr	Columns[RF]
	00 00 00 08 00 06 40 02	"ACE_K_AES_256_GlobalRange_Set_PendingAccessConditions"	See note 2	Admins[WF]	PendingAccessConditions
	00 00 00 08 00 06 41 01	"ACE_K_AES_256_Range1_Set_PendingAccessConditions"	See note 2	Admins[WF]	PendingAccessConditions
	00 00 00 08 00 06 41 00 (+NNNN)	"ACE_K_AES_256_RangeNNNN_Set_PendingAccessConditions"	See note 2	Admins[WF]	PendingAccessConditions
AccessCondition					
	00 00 00 08 00 06 42 01	"ACE_AccessCondition_AccessCondition1_Set_DisableOnReset"	See note 2	Admins[WF]	DisableOnReset
	00 00 00 08 00 06 42 00 (+XXXX)	"ACE_AccessCondition_AccessConditionXXXX_Set_DisableOnReset"	See note 2	Admins[WF]	DisableOnReset

Table Association -Informative Column	UID[RF]	Name[1]	CommonName	BooleanExpr	Columns[RF]
	00 00 00 08 00 06 43 01	"ACE_AccessCondition_AccessCondition1_InitializeAccessCondition"	See note 2	Admins[WF]	All
	00 00 00 08 00 06 43 00 (+XXXX)	"ACE_AccessCondition_AccessConditionXXXX_InitializeAccessCondition"	See note 2	Admins[WF]	All
	00 00 00 08 00 06 44 01	"ACE_AccessCondition_AccessCondition1_TestAccessKey"	See note 2	Admins[WF]	All
	00 00 00 08 00 06 44 00 (+XXXX)	"ACE_AccessCondition_AccessConditionXXXX_TestAccessKey"	See note 2	Admins[WF]	All
	00 00 00 08 00 06 45 01	"ACE_AccessCondition_AccessCondition1_ChangeAccessKey"	See note 2	Admins[WF]	All
	00 00 00 08 00 06 45 00 (+XXXX)	"ACE_AccessCondition_AccessConditionXXXX_ChangeAccessKey"	See note 2	Admins[WF]	All

Table Association -Informative Column	UID[RF]	Name[1]	CommonName	BooleanExpr	Columns[RF]
	00 00 00 08 00 06 46 01	"ACE_AccessCondition_AccessCondition1_EnableAccess"	See note 2	Admins[WF]	All
	00 00 00 08 00 06 46 00 (+XXXX)	"ACE_AccessCondition_AccessConditionXXXX_EnableAccess"	See note 2	Admins[WF]	All
	00 00 00 08 00 06 47 01	"ACE_AccessCondition_AccessCondition1_DisableAccess"	See note 2	Admins[WF]	All
	00 00 00 08 00 06 47 00 (+XXXX)	"ACE_AccessCondition_AccessConditionXXXX_DisableAccess"	See note 2	Admins[WF]	All

Notes:

1. The text in each table cell in the column is informative. See the TCG Core Specification [2] and the applicable TCG Opal Family SSC.
2. The content of the cell is defined by the TCG Core Specification [2] and the applicable TCG Opal Family SSC.
3. RF: For table cells, the content of the cell is Read-only as specified by the applicable TCG Opal Family SSC. For table columns, the content of each cell in the column is Read-only as specified by the applicable TCG Opal Family SSC.
4. WF: The content of the cell is writable as specified by section 5.5.1.1.3, but the access control is fixed as specified by the applicable TCG Opal Family SSC.

5.5.1.1.5 K_AES_256

The K_AES_256 Table is defined in [2]. In addition to the requirements in [3], the K_AES_256 Table Preconfiguration data SHALL be modified as specified in Table 14.

Table 14: LockingSP - K_AES_256 Table Preconfiguration

UID[RF]	Name[1]	ActiveAccessConditions[WP]	PendingAccessConditions[WP]
00 00 08 06 00 00 00 01	"K_AES_256_GlobalRange_Key"	[ ]	[ ]
00 00 08 06 00 03 00 01	"K_AES_256_Range1_Key"	[ ]	[ ]
00 00 08 06 00 03 NN NN	"K_AES_256_RangeNNNN_Key"	[ ]	[ ]

Notes:
1. The text in each table cell in the column is informative. See the TCG Core Specification [2] and the applicable TCG Opal Family SSC.
2. RF: The content of each cell in the column is Read-only as specified by the applicable TCG Opal Family SSC.
3. WP: The content of each cell in the column is writeable (e.g., ActiveAccessConditions column is indirectly writeable using the GenKey method as specified by section 4.2.2.2.1). The access control is personalizable as specified by section 5.5.1.1.3.

5.5.1.1.6 AccessCondition

Table 15 shows the Preconfiguration data for the new AccessCondition table.

Table 15: LockingSP - AccessCondition Table Preconfiguration

UID[RF]	Name[1]	CommonNames	AccessEnabled[WP]	DisableAccessOnReset[WP]	EncapsulationKeys[RFV]
00 00 17 01 00 00 00 01	"AccessCondition1"	See note 5	False	PowerCycle	VU
00 00 17 01 00 00 00 00 (+XXXX) (See note 6)	"AccessConditionXXXX"	See note 5	False	PowerCycle	VU

UID[RF]	Name[1]	CommonNames	AccessEnabled[WP]	DisableAccessOnReset[WP]	EncapsulationKeys[RFV]

Notes:
1. The text in each table cell in the column is informative. See the TCG Core Specification [2] and the applicable TCG Opal Family SSC.
2. RF: The content of each cell in the column is Read-only as specified by section 5.5.1.1.3.
3. WP: The content of each cell in the column is indirectly writeable using the `EnableAccess` or `DisableAccess` methods and the associated access control is personalizable as specified by section 5.5.1.1.3
4. RFV: The content of each cell in the column is Read-only as specified by section 5.5.1.1.3. The value of the cell is vendor unique.
5. The content of the cell is defined by the TCG Core Specification [2] and the applicable TCG Opal Family SSC.
6. `AccessCondition` Table's objects are added to the table; where XXXX can range from 2 to the maximum value supported by the TPer as reported in section 5.2.1.5.

5.5.1.1.6.1 DisableAccessOnReset Restrictions

The TPer SHALL support the following DisableAccessOnReset column values:

a) { 0 } (i.e., Power Cycle); and

b) { 0, 3 } (i.e., Power Cycle and Programmatic).

Additionally, the TPer MAY support the following DisableAccessOnReset column values:

a) { 0, 1 } (i.e., Power Cycle and Hardware Reset); and

b) { 0, 1, 3 } (i.e., Power Cycle, Hardware Reset, and Programmatic).

5.5.1.1.7 Certificates

The `Certificates` Table is defined in [2]. In addition to requirements in [2], the `Certificates` Table Preconfiguration data SHALL be modified as specified in Table 16.

Table 16: LockingSP - Certificates Table Preconfiguration

UID[RF]	Name[1]	CommonNames	CertData[WF]	CertSize[WF]	EphemeralKeyPair	NextCertificate
00 00 00 0A 00 00 04 01	"Certificate1"	See note 5	VU	VU	True[RF]	VU[RFV]
00 00 00 0A 00 00 04 00 (+MMMM) (See note 6)	"CertificateMMMM"	See note 5	VU	VU	VU[RFV]	VU[RFV]

Notes:
1. The text in each table cell in the column is informative. See the TCG Core Specification [2] and the applicable TCG Opal Family SSC.
2. RF: For table cells, the content of the cell is Read-only as specified by section 5.5.1.1.3. For table columns, the content of each cell in the column is Read-only as specified by section 5.5.1.1.3.
3. WF: The content of each cell in the column is indirectly writeable when EphemeralKeyPair column value is set to True using `GenKey` method or other events specified in section 4.2.2.1.1. CertData column value is not writable if the EphemeralKeyPair column value is set to False (see 4.2.2.1.1 for details). The access control is fixed as specified by the applicable TCG Opal Family SSC.
4. RFV: The content of the cell is Read-only as specified by section 5.5.1.1.3. The value of the cell is vendor unique.
5. The content of the cell is defined by the TCG Core Specification [2] and the applicable TCG Opal Family SSC.
6. `Certificates` Table's objects are added to the table; where MMMM can range from 2 to the maximum number of certificates supported by the TPer as described in section 4.2.2.1.3.

5.5.1.1.8 CertificateData

The `CertificateData` table is a byte table that contains the actual certificate data of the `Certificates` table. Its size is dependent on the X.509 DER certificate object that it contains as defined by the Storage Device manufacturer. The lifecycle of the contents of this table are tied to the lifecycle of the certificate objects that the table contains as specified in section 4.2.2.1.1

5.6 Additional SPs

This feature set requires no additional SPs.

5.7 Single User Mode Feature Set Interactions

5.7.1 Overview

The Single User Mode Feature Set (see [8]) MAY be supported on a TPer that supports the MEK Multiparty Authorization Feature Set.

This section describes the interactions when the MEK Multiparty Authorization Feature Set and the Single User Mode Feature Set (see [8]) are both supported and present (enabled or disabled).

5.7.2 Modified Methods

This section defines modifications to methods that are required to support the Single User Mode Feature Set (see [8]) in addition to the MEK Multiparty Authorization Feature Set.

5.7.2.1 Reactivate

`Reactivate` method interactions with MEK Multiparty Authorization are that when the `Reactivate` method defined in [8] is successfully invoked, the TPer SHALL modify all objects in the `Certificates` table and the corresponding `CertificateData` table where the EphemeralKeyPair column is True as specified in section 4.2.2.1.1.

All PendingAccessConditions column values if configured are preserved across the method invocation.

Start of Informative Comment

As specified in [8] , the `Reactivate` method requires all `Locking` table objects to have ReadLockEnabled and WriteLockEnabled column values to be False prior to successful invocation. Therefore, the ActiveAccessConditions column of all `Locking` table objects' `K_AES_256` table objects are necessarily empty lists.

Since media encryption keys are preserved across the transition, the PendingAccessConditions column can be non-empty without conflicting with the method requirements.

Note that disabling locking for all `Locking` table objects (i.e., setting their ReadLockEnabled and WriteLockEnabled column values to be False) can only be accomplished when those `Locking` table objects are not associated with any AccessCondition object.

This implies that it's not possible to preserve host user data across `Reactivate` invocation if any of the `Locking` table objects being activated in Single User mode is already associated with an AccessCondition object. A host wishing to preserve host data bound to AccessCondition protection needs to ensure that `Reactivate` is invoked prior to configuring AccessCondition objects for the `Locking` table objects intended to be activated in Single User mode.

End of Informative Comment

When the `Reactivate` method defined in [8] is invoked:

- If the SingleUserModeSelectionList parameter is included and specifies ranges associated with a `K_AES_256` table object containing non-empty ActiveAccessConditions list, the method SHALL fail with a status of INVALID_PARAMETER.
- If AccessCondition objects in the ActiveAccessConditions column associated with the specified ranges have an AccessEnabled column value of False, the method SHALL also fail with a status of INVALID_PARAMETER. (Check the Activate in SUM as ActiveAccessConditions being non-empty)

5.7.2.2 Erase

Start of Informative Comment

Per [8], the purpose of the `Erase` method is to cryptographically erase user data within a specific LBA Range and to reset the locking states of that LBA Range, as well as the C_PIN column value of the User authority associated with that range. The method requires that a new media encryption key be generated for the range during the processing.

The MEK Multiparty Authorization feature set requires that media encryption keys be constrained by an immutable set of AccessCondition objects for the lifetime of those keys. Further, when generating new media encryption keys for `K_AES_256` table object, they are expected to be constrained by the AccessCondition objects in the PendingAccessConditions list as described in 4.2.2.2.2.

As described in 4.2.2.2.1, a `K_AES_256` table object with a non-empty ActiveAccessConditions or non-empty PendingAccessConditions list are expected to have associated ReadLockEnabled and WriteLockEnabled values of True to guarantee AccessCondition-backed data-at-rest protections for the range across resets.

End of Informative Comment

If:

- the `Erase` method defined in [8] is successfully invoked on a Locking object;
- the PendingAccessConditions column of the `K_AES_256` table object referenced in the ActiveKey column is not an empty list; and
- all AccessCondition object in the PendingAccessConditions list have an AccessEnabled column value of True;

Then:

- The ReadLockEnabled and WriteLockEnabled column values of the Locking object SHALL be set to True.

- The ActiveAccessConditions column of the `K_AES_256` table object SHALL be set to the value of the PendingAccessConditions column.
- A new media encryption key value SHALL be generated after the ActiveAccessConditions column is updated.
- The usage of the new media encryption key SHALL be constrained as defined by the ActiveAccessConditions column (see section 4.2.1.1.4).
- The PendingAccessConditions column value SHALL be preserved across the method invocation.

Start of Informative Comment

To have media encryption keys correctly protected by the AccessCondition objects in the PendingAccessConditions column, they are expected to have been previously enabled. Therefore, the Erase method is constrained as defined in 4.2.2.2.2.1.

End of Informative Comment

If:

- the `Erase` method defined in [8] is successfully invoked on a Locking object;
- the PendingAccessConditions column of the `K_AES_256` table object referenced in the ActiveKey column is not an empty list; and
- any AccessCondition object in the PendingAccessConditions list has an AccessEnabled column value of False;

Then the method SHALL fail with a status of FAIL.

5.8 Configurable Namespace Locking Feature Set Interactions

5.8.1 Overview

The Configurable Namespace Locking Feature Set (see [9]) MAY be supported on a TPer that supports the MEK Multiparty Authorization Feature Set.

This section describes the interactions when the MEK Multiparty Authorization Feature Set and the Configurable Namespace Locking Feature Set (see [9]) are both supported and present (enabled or disabled).

5.8.2 Interactions with Globally-Associated Namespaces

All namespaces that are associated with the Global Range Locking object SHALL be constrained by `AccessCondition` objects associated with the Global Range Locking object as specified in section 4.2.1.1.4.

5.8.3 Modified Methods

This section defines modifications to methods that are required to support the Configurable Namespace Locking Feature Set (see [9]) in addition to the MEK Multiparty Authorization Feature Set.

5.8.3.1 Assign

`Assign` method interactions with MEK Multiparty Authorization are defined in the following subsections.

5.8.3.1.1 Assigning a Namespace Global Range Locking object

Start of Informative Comment

The MEK Multiparty Authorization feature set requires that a given Media Encryption Key remains bound by a fixed set of AccessCondition objects throughout its lifecycle. The following specifies that any AccessCondition objects associated with Global Range are transferred along with the Media Encryption Key of the namespace to the Namespace Global Range to preserve that invariant.

Additionally, the Assign method requires that the Global Range Locking object have WriteLocked and ReadLocked values of False to configure the Namespace Global Range Locking object for a namespace. If any AccessCondition objects are configured for the Global Range, this necessarily implies that all AccessCondition objects in the ActiveAccessConditions column of the K_AES_256 table object indicated by the ActiveKey column of the Global Range Locking object have an AccessEnabled value of True; otherwise, the Locking object are expected to have a value of True in both WriteLocked and ReadLocked columns (see section 4.2.2.2.1.1 for details).

End of Informative Comment

If Assign is used to configure the Namespace Global Range Locking object, then the TPer SHALL perform the following additional actions:

- the ActiveAccessConditions and PendingAccessConditions objects from the K_AES_256 table object of the Global Range Locking object SHALL be copied to the K_AES_256 table object indicated by the ActiveKey column of the Locking table object selected to become the Namespace Global Range Locking object.
- If the ActiveAccessConditions list copied is non-empty, the TPer SHALL transition the ReadLockEnabled and WriteLockEnabled columns of the selected Locking table object to True.
- any Media Encryption Key previously associated with the K_AES_256 table object indicated by the selected Locking table object SHALL be eradicated.

5.8.3.1.1.1 Interactions with Single User Mode Feature Set

Start of Informative Comment

The following clarifies that new Media Encryption Keys generated as a result of the Assign method will inherit any ActiveAccessConditions bindings from the Global Range ActiveAccessConditions when the Global Range is in Single User Mode.

End of Informative Comment

If:

a) the K_AES_256 table object indicated by the ActiveKey column value of the Global Range Locking object has a non-empty list in the ActiveAccessConditions column or PendingAccessConditions column;

b) the UID of the Global Range Locking object is included in the single_user_ranges list in the SingleUserModeRanges column of the LockingInfo table; and

c) the RangeStartLengthPolicy column value of the LockingInfo table equals to one

Then:

a) the ActiveAccessConditions and PendingAccessConditions of the K_AES_256 table object of the Global Range Locking table object SHALL be copied to the selected Locking object as described in 5.8.3.1.1;
b) if the ActiveAccessConditions list copied is non-empty, the TPer SHALL transition the ReadLockEnabled and WriteLockEnabled columns of the selected Locking table object SHALL be set to True; and
c) the new MEK generated for the selected Locking table object SHALL be constrained by the associated AccessCondition objects as defined in section 4.2.1.1.4

5.8.3.1.2 Assigning a Namespace Non-Global Range Locking object

If Assign is used to configure a Namespace Non-Global Range Locking object, then the following additional actions occur:

a) the ActiveAccessConditions and PendingAccessConditions from the K_AES_256 table object of the Namespace Global Range Locking object SHALL be copied to the K_AES_256 table object referenced in the ActiveKey column of the Locking table object selected to become the Namespace Non-Global Range Locking object;
b) if the ActiveAccessConditions list copied is non-empty, the TPer SHALL transition the ReadLockEnabled and WriteLockEnabled columns of the selected Locking table object to True;
c) the new MEK generated for the selected Locking table object SHALL be constrained by the associated AccessCondition objects as defined in section 4.2.1.1.4; and
d) any Media Encryption Key previously associated with the K_AES_256 table object indicated by the select Locking table object SHALL be eradicated.

5.8.3.2 Deassign

Deassign method interactions with MEK Multiparty Authorization are defined in the following subsections.

5.8.3.2.1 Deassigning a Namespace Global Range

Start of Informative Comment

The following clarifies that any new media encryption keys generated as a result of the Deassign method on a Namespace Global Range will be constrained by the ActiveAccessConditions of the Global Range Locking object.

End of Informative Comment

If:

a) The ActiveAccessConditions column of the K_AES_256 table object indicated by the ActiveKey column of the Global Range Locking object is not an empty list; and
b) the KeepNamespaceGlobalRangeKey parameter:
    a. is False; or
    b. is not specified;

then the Deassign method:

a) SHALL succeed as specified in [9].
b) SHALL constrain any new media encryption key generated for the Namespace with the ActiveAccessConditions of the K_AES_256 table object indicated by the ActiveKey column of the Global Range Locking object (see section 4.2.2.2.1); and
c) SHALL reset the ActiveAccessConditions and PendingAccessConditions columns values of the K_AES_256 object indicated by the ActiveKey column of the Locking object indicated by the UID parameter to its preconfigured default (i.e., original factory state).

Start of Informative Comment

The MEK Multiparty Authorization feature set requires that a media encryption key always be associated with the same set of AccessCondition objects during its lifetime. The following clarifies that setting the KeepNamespaceGlobalRangeKey parameter to True is only valid if the ActiveAccessConditions list of the target Locking table object matches the Global Range Locking object ActiveAccessConditions, such that the invariant condition is maintained.

End of Informative Comment

If:

a) the Locking object indicated by the UID parameter has the NamespaceGlobalRange column value set to True;

b) the KeepNamespaceGlobalRangeKey parameter is True; and
c) the ActiveAccessConditions column of the K_AES_256 table object indicated by the ActiveKey column of the Global Range Locking object does not match the ActiveAccessConditions column of the K_AES_256 table object indicated by the ActiveKey column of the Locking object indicated by the UID parameter.

Then the method SHALL fail with a status of INVALID_PARAMETER.

If:

a) the Locking object indicated by the UID parameter has the NamespaceGlobalRange column value set to True;
b) the KeepNamespaceGlobalRangeKey parameter is True; and
c) the ActiveAccessConditions column of the K_AES_256 table object indicated by the ActiveKey column of the Global Range Locking object match the ActiveAccessConditions column of the K_AES_256 table object indicated by the ActiveKey column of the Locking object indicated by the UID parameter.

Then the Deassign method:

a) SHALL succeed as specified in [9]; and
b) SHALL reset the ActiveAccessConditions and PendingAccessConditions columns values of the K_AES_256 table object indicated by the ActiveKey column of Locking object indicated by the UID parameter to its preconfigured default (i.e., original factory state).

5.8.3.2.2 Deassigning a Namespace Non-Global Range

Start of Informative Comment

The following clarifies that LBA range returned to the Namespace Global Range Locking object as a result of the Deassign method on a Namespace Non-Global Range Locking object will be constrained by the AccessCondition objects associated with the Namespace Global Range Locking object.

End of Informative Comment

If:

a) the Locking object indicated by the UID parameter has the NamespaceGlobalRange column value set to False; and
b) the ActiveAccessConditions column of the K_AES_256 table object indicated by the ActiveKey column of the Namespace Global Range Locking object associated with LBA range being deassigned is not empty.

Then the Deassign method:

a) SHALL succeed as specified in [9].
b) SHALL reset the ActiveAccessConditions and PendingAccessConditions columns values of the K_AES_256 table object indicated by the ActiveKey column of Locking object indicated by the UID parameter to its preconfigured default; and
c) SHALL constrain the LBA range returned to the associated Namespace Global Range Locking object with the ActiveAccessConditions of the K_AES_256 table object indicated by the ActiveKey column of the Namespace Global Range Locking object (see section 4.2.2.2.1).