TRUSTED® COMPUTING GROUP

SPECIFICATION

TCG Storage Security Subsystem Class (SSC): Key Per I/O

Version 1.00
Revision 1.41
September 1, 2023

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 the industry at large.

NAME	COMPANY
Burt Wagner	CNEX Labs, Inc.
Chandra Nelogal	Dell Technologies, Inc.
Kevin Marks	Dell Technologies, Inc.
Chris Hillier	HPE
Glen Jacquette	IBM
Jacob Sheppard	IBM
Alan Bumgarner	Intel, Solidigm
Festus Hategekimana	Intel, Solidigm
Michal Antoniewski	Intel, Solidigm
Thomas Bowen	Intel
James Borden	Kioxia Corporation
John Geldman	Kioxia Corporation
Mark Carlson	Kioxia Corporation
Paul Suhler	Kioxia Corporation
Taku Kato	Kioxia Corporation
Alan Arnold	Lenovo Inc.
Ke Du	Marvell Technology
Michael McDonnell	Marvell Technology
Steve Yu	Marvell Technology
Alon Cohen	Microchip Technology
Artem Zankovich	Micron Technology, Inc.
Bharath Madanayakanahali Gururaj	Micron Technology, Inc.
Robert Strong	Micron Technology, Inc.
Walt Hubis	Micron Technology, Inc.
Fred Knight	NetApp
Sridhar Balasubramanian	NetApp
Tim Chevalier	NetApp
Cheri Hawkins	NSA
Uyen Nguyen	NSA
Dick Wilkins	Phoenix Technologies
Caroline Kahn	Samsung Semiconductor Inc.
Eric Hibbard	Samsung Semiconductor Inc.
Judy Brock	Samsung Semiconductor Inc.
Anthony Duran	Seagate Technology
Jim Hatfield	Seagate Technology
Dharma Nagarajan	SK Hynix
Han Choi	SK Hynix
Santosh Kumar	SK Hynix
John Mathews	Solidigm
Patrick Hery	Toshiba Corporation
Joseph Chen	ULINK Technology Inc.
Cyril Guyot	Western Digital Technologies, Inc.
Danny Barra	Western Digital Technologies, Inc.
Dave Landsman	Western Digital Technologies, Inc.

Jithendra Bethur	Western Digital Technologies, Inc.
Raj Bhagwat	Western Digital Technologies, Inc.
Yoni Shternhell	Western Digital Technologies, Inc.
David Challener	David Challener

CONTENTS

1 INTRODUCTION 13

1.1 Document Purpose 13
1.2 Scope and Intended Audience 13
1.3 Conventions 13
1.3.1 Key Words 13
1.3.2 Font Conventions 13
1.3.3 Statement Types 13
1.3.4 SP Table Cell Color Legend 14
1.3.5 List Conventions 15
1.3.5.1 Lists Overview 15
1.3.5.2 Unordered Lists 15
1.3.5.3 Ordered Lists 15
1.3.6 Numbering 16
1.3.7 Bit Conventions 16
1.3.8 Number Range Convention 16
1.3.9 Specify and Provide Convention 16
1.4 Document References 17
1.4.1 Document Precedence 17
1.4.2 Approved References 17
1.5 Dependencies on Other Feature Sets 17
1.6 Interactions with Other Feature Sets 18
1.7 Definition of Terms 18

2 KEY PER I/O SSC OVERVIEW 21

2.1 Key Per I/O SSC Use Cases and Threats 21
2.2 Provisioning Keys into a Storage Device 23
2.2.1 Example Diagram Conventions 24
2.2.2 Key Generation 24
2.2.3 KEK Provisioning Models 25
2.2.3.1 Static Pre-shared Plaintext Key Encryption Keys 26
2.2.3.1.1 Provisioning Plaintext Key Encryption Keys 26
2.2.3.2 PKI-based Key Encryption Keys Transport Mechanism 27
2.2.3.2.1 KEK Provisioning Using Key Transport scheme such as KTS-OAEP 28
2.2.3.3 Updating Key Encryption Keys (KEKs) Using a Previously Provisioned KEK 31
2.2.4 Media Encryption Key (MEK) Injection 32
2.2.5 Key Selection Theory of Operation 34
2.3 Security Providers (SPs) 36
2.4 Interface Communication Protocol 36
2.5 Cryptographic Features 37
2.6 Authentication 37
2.7 Table Management 37
2.8 Access Control & Personalization 37
2.9 Issuance 37
2.10 Key Per I/O SSC Discovery 37
2.11 Mandatory Feature Sets 37

3 KEY PER I/O SSC FEATURES 38

3.1 SECURITY PROTOCOL 1 SUPPORT 38
3.1.1 Level 0 Discovery (M) 38
3.1.1.1 Level 0 Discovery Header 38
3.1.1.2 TPer Feature (Feature Code = 0x0001) 39
3.1.1.3 Key Per I/O SSC v1.00 Feature (Feature Code = 0x0305) 40
3.1.1.3.1 Protocol 0x01 Base ComID 41
3.1.1.3.2 Protocol 0x01 Number of ComIDs 41
3.1.1.3.3 Protocol 0x03 Base ComID 41
3.1.1.3.4 Protocol 0x03 Number of ComIDs 42
3.1.1.3.5 Initial C_PIN_SID PIN Indicator 42
3.1.1.3.6 Behavior of C_PIN_SID PIN upon TPer Revert 42
3.1.1.3.7 Number of Key Per I/O SP Admin Authorities 42
3.1.1.3.8 Key Per I/O Enabled 42
3.1.1.3.9 Key Per I/O Scope 42
3.1.1.3.10 Shared XTS-AES Tweak Key Required 43
3.1.1.3.11 Incorrect Key Detection Supported 43
3.1.1.3.12 Replay Protection Supported 43
3.1.1.3.13 Replay Protection Enabled 43
3.1.1.3.14 Maximum Supported Key Unique Identifier Length 44
3.1.1.3.15 KMIP Formatted Key Injection Supported 44
3.1.1.3.16 NIST AES-KW Supported 44
3.1.1.3.17 NIST AES-GCM Supported 45
3.1.1.3.18 NIST RSA-OAEP Supported 45
3.1.1.3.19 AES-256 Wrapping Key Supported 45
3.1.1.3.20 RSA2K Wrapping Key Supported 45
3.1.1.3.21 RSA3K Wrapping Key Supported 45
3.1.1.3.22 RSA4K Wrapping Key Supported 46
3.1.1.3.23 Plaintext Key Encryption Keys Provisioning Supported 46
3.1.1.3.24 PKI-based Key Encryption Keys Transport Supported 46
3.1.1.3.25 Number of Key Encryption Keys Supported 46
3.1.1.3.26 Total Number of Key Tags Supported 46
3.1.1.3.27 Maximum Number of Key Tags Supported Per Namespace 46
3.1.1.3.28 Get Nonce Command Nonce Length 47
3.1.1.3.29 Key Per I/O SSC Feature Descriptor Requirements 47
3.1.1.4 Supported Data Removal Mechanism Feature (Feature Code = 0x0404) 49
3.1.1.4.1 Data Removal Operation Processing Definition 50
3.1.1.4.2 Data Removal Operation Interrupted 50
3.1.1.4.3 Supported Data Removal Mechanism Definition 50
3.1.1.4.4 Data Removal Time Format and Data Removal Time Definition 51
3.1.2 Namespace Level 0 Discovery 52
3.1.2.1 Overview 52
3.1.2.2 IF-SEND Command 52
3.1.2.3 IF-RECV Command 52
3.1.2.3.1 Length of Parameter Data 53
3.1.2.3.2 Data Structure Revision 54
3.1.2.4 Namespace Level 0 Discovery Feature Descriptors 54
3.1.2.5 Namespace Key Per I/O Capabilities Feature (Feature Code = 0x040A) 54
3.1.2.5.1 Namespace Key Per I/O Capabilities Feature Code 54
3.1.2.5.2 Version 54
3.1.2.5.3 Length 55

3.1.2.5.4 Managed By Key Per I/O 55
3.1.2.5.5 Number of Allocated Key Tags 55
3.1.2.5.6 Namespace Key Per I/O Capabilities Feature Descriptor Requirements 55
3.2 SECURITY PROTOCOL 2 SUPPORT 56
3.2.1 ComID Management *56*
3.2.2 Stack Protocol Reset (M) *56*
3.2.3 TPER_RESET command (M) *56*
3.2.4 Clear Single MEK Command (M) *56*
3.2.4.1 Command Overview 56
3.2.5 Clear All MEKs command (M) *59*
3.2.5.1 Command Overview 59
3.2.6 Get Nonce Command (O) *61*
3.3 SECURITY PROTOCOL 3 SUPPORT 62
3.3.1 TPer – Host Communication Over Protocol ID 0x03 ..... *63*
3.3.2 ComID Management *64*
3.3.3 ComID Binding to Security Protocol *64*
3.3.3.1 TCG Resets and ComID Binding 64
3.3.3.2 Security Protocol 0x02 and ComID Binding 64
3.3.4 Security Protocol 3 Payload *65*
3.4 COMMUNICATIONS 66
3.4.1 Communication Properties *66*
3.4.2 Supported Security Protocols *67*
3.4.3 ComIDs *67*
3.4.4 Synchronous Protocol *67*
3.4.4.1 Payload Encoding 67
3.4.4.1.1 Stream Encoding Modifications 67
3.4.4.1.2 TCG Packets 68
3.4.4.1.3 Payload Error Response 68
3.4.5 Storage Device Resets *69*
3.4.5.1 Interface Resets 69
3.4.5.2 TCG Reset Events 69
3.4.6 Protocol Stack Reset Commands (M) *69*

4 KEY PER I/O SSC-COMPLIANT FUNCTIONS AND SPS 70

4.1 SESSION MANAGER 70
4.1.1 Methods *70*
4.1.1.1 Properties (M) 70
4.1.1.1.1 Protocol3MaxPayloadSize 71
4.1.1.1.2 Protocol3MaxKmipBatchItems 72
4.1.1.2 StartSession (M) 72
4.1.1.3 SyncSession (M) 72
4.1.1.4 CloseSession (O) 72
4.2 ADMIN SP 72
4.2.1 Base Template Tables *72*
4.2.1.1 SPInfo (M) 72
4.2.1.2 SPTemplates (M) 73
4.2.1.3 Table (M) 73
4.2.1.4 MethodID (M) 74
4.2.1.5 AccessControl (M) 75
4.2.1.6 ACE (M) 81

4.2.1.7 Authority (M) 83
4.2.1.8 C_PIN (M) 83
4.2.2 Base Template Methods *84*
4.2.3 Admin Template Tables *84*
4.2.3.1 TPerInfo (M) 84
4.2.3.2 Template (M) 85
4.2.3.3 SP (M) 85
4.2.4 Admin Template Methods *86*
4.2.5 Key Per I/O SSC Additional Column Types *86*
4.2.5.1 Data_removal_mechanism 86
4.2.6 Key Per I/O SSC Additional Data Structures *86*
4.2.6.1 DataRemovalMechanism (Object Table) 86
4.2.6.1.1 UID 87
4.2.6.1.2 ActiveDataRemovalMechanism 87
4.2.7 Key Per I/O SSC Additional Tables *87*
4.2.7.1 DataRemovalMechanism (M) 87
4.2.8 Crypto Template Tables *87*
4.2.9 Crypto Template Methods *87*
4.2.9.1 Random 87
4.3 KEY PER I/O SP 87
4.3.1 Base Template Tables *87*
4.3.1.1 SPInfo (M) 88
4.3.1.2 SPTemplates (M) 88
4.3.1.3 Table (M) 88
4.3.1.4 Type (N) 89
4.3.1.4.1 New kek_obj_uidref type 89
4.3.1.4.2 New kek_obj_uidref_list type 90
4.3.1.4.3 New KeyEncryptionKey_Type type 90
4.3.1.5 MethodID (M) 90
4.3.1.6 AccessControl (M) 91
4.3.1.7 ACE (M) 100
4.3.1.8 Authority (M) 102
4.3.1.9 C_PIN (M) 103
4.3.1.10 Certificates (O) 103
4.3.2 Base Template Methods *104*
4.3.3 Crypto Template Tables *104*
4.3.4 Crypto Template Methods *104*
4.3.4.1 Random 104
4.3.5 SSC Specific Tables *104*
4.3.5.1 KPIOPolicies (M) 104
4.3.5.1.1 UID 104
4.3.5.1.2 ClearSingleMEKAllowed 105
4.3.5.1.3 ClearAllMEKsAllowed 105
4.3.5.1.4 ReplayProtectionEnabled 105
4.3.5.1.5 PlaintextKEKProgrammingEnabled 105
4.3.5.1.6 PKIProtectedKEKProgrammingEnabled 106
4.3.5.1.7 KeyInjectionInterfaceLockEnabled 107
4.3.5.1.8 KeyInjectionInterfaceLocked 107
4.3.5.1.9 KeyInjectionInterfaceLockOnReset 108
4.3.5.1.10 KPIOPolicies Table Preconfiguration 108
4.3.5.1.11 KeyInjectionInterfaceLockOnReset Restrictions 108

4.3.5.2 KeyTagAllocation (M) 109
4.3.5.2.1 UID 109
4.3.5.2.2 Name 109
4.3.5.2.3 CommonName 109
4.3.5.2.4 NamespaceID 109
4.3.5.2.5 Managed 109
4.3.5.2.6 NumberOfKeyTags 110
4.3.5.2.7 AllowedKeyEncryptionKeys 111
4.3.5.2.8 KeyTagAllocation Table Preconfiguration 111
4.3.5.3 KeyEncryptionKey (M) 112
4.3.5.3.1 UID 113
4.3.5.3.2 Name 113
4.3.5.3.3 CommonName 113
4.3.5.3.4 AccessLockEnabled 113
4.3.5.3.5 AccessLocked 113
4.3.5.3.6 LockOnReset 113
4.3.5.3.7 AllowedKeyEncryptionKeys 114
4.3.5.3.8 KMIP KeyUID 114
4.3.5.3.9 Key 114
4.3.5.3.10 KeyEncryptionKey Table Preconfiguration 114
4.3.5.3.11 LockOnReset Restrictions 116
4.3.5.4 KeyPerIOPublicKeyCertificateData (O) 116

5 APPENDIX – KEY PER I/O SSC SPECIFIC FEATURES 117

5.1 KEY PER I/O SSC-SPECIFIC METHODS 117
5.1.1 Activate – Admin Template SP Object Method 117
5.1.1.1 Activate Support 117
5.1.1.2 Side effects of Activate 117
5.1.2 Revert – Admin Template SP Object Method 118
5.1.2.1 Revert Support 118
5.1.2.2 Side effects of Revert 118
5.1.2.2.1 Effects of Revert on the PIN Column Value of C_PIN_SID 119
5.2 LIFE CYCLE 120
5.2.1 Issued vs Manufactured SPs 120
5.2.1.1 Issued SPs 120
5.2.1.2 Manufactured SPs 120
5.2.2 Manufactured SP Life Cycle States 120
5.2.2.1 State definitions for Manufactured SPs 120
5.2.2.2 State transitions for Manufactured SPs 121
5.2.2.2.1 Manufactured-Inactive to Manufactured Transition 121
5.2.2.2.2 ANY STATE to ORIGINAL FACTORY STATE Transition 121
5.2.2.3 State behaviors for Manufactured SPs 121
5.2.3 LifeCycle Type Modification 122
5.3 BYTE TABLE ACCESS GRANULARITY 123
5.3.1 Table Table Modification 123
5.3.1.1 MandatoryWriteGranularity 123
5.3.1.1.1 Byte Tables 123
5.3.1.1.2 Object Tables 124
5.3.1.2 RecommendedAccessGranularity 124
5.3.1.2.1 Byte Tables 124
5.3.1.2.2 Object Tables 125

5.4 MAPPING BETWEEN KEY PER I/O AND OASIS KMIP 125
5.4.1 Minimum KMIP Specification Version *126*
5.4.2 Required KMIP Capabilities *126*
5.4.2.1 KMIP Communication Protocol 126
5.4.2.2 KMIP Objects 126
5.4.2.3 KMIP Operations 127
5.4.2.4 KMIP Operations Data Structures 127
5.4.2.5 KMIP Object Attributes 128
5.4.3 Key Per I/O and OASIS KMIP Terminology Mapping *128*
5.4.4 Key Per I/O Storage Device Interactions with KMIP *130*
5.4.4.1 Interactions with KMIP Request Message and Response Message Data Structures 130
5.4.4.1.1 Request Message Header 130
5.4.4.2 Key Injections Interactions 132
5.4.4.2.1 KMIP Key Injection Interface Locking 132
5.4.4.2.2 Interactions with KMIP Symmetric Key's Object Attributes 132
5.4.4.2.3 Key Encryption Keys Injection 140
5.4.4.2.4 Media Encryption Keys Injection 146
5.4.4.2.5 Response Message Considerations 152
5.4.4.3 Discovering Supported KMIP Capabilities 154
5.4.4.3.1 Supported Versions 154
5.4.4.3.2 Supported KMIP Operations and Object Types 154
5.5 INTERFACE READ AND WRITE INTERACTIONS 154
5.5.1 Reading User Data *154*
5.5.2 Writing User Data *155*
5.6 INTERACTIONS WITH RESETS 155
5.6.1 Interactions with Power Cycle *155*
5.7 INTERACTIONS WITH NVME NAMESPACE MANAGEMENT 155
5.8 INTERACTIONS WITH THE FORMAT NVM COMMAND 155
5.9 INTERACTIONS WITH THE SANITIZE COMMAND 155
5.10 INTERACTIONS WITH THE LOCKING TEMPLATE 156
5.11 INTERACTIONS WITH THE IDENTIFY NAMESPACE DATA STRUCTURE 156
5.12 INTERACTIONS WITH THE IDENTIFY CONTROLLER DATA STRUCTURE 156

List of Tables

Table 1: SP Table Legend 14
Table 2: Level 0 Discovery Header 38
Table 3: Level 0 Discovery - TPer Feature Descriptor 39
Table 4: Level 0 Discovery – Key Per I/O SSC Feature Descriptor 40
Table 5: SSC Minor Versions 47
Table 6: Level 0 Discovery – Supported Data Removal Mechanism Feature Descriptor 49
Table 7: Supported Data Removal Mechanism 51
Table 8: Data Removal Time (Data Removal Time Format bit= 0) 52
Table 9: Data Removal Time (Data Removal Time Format bit= 1) 52
Table 10: Namespace Level 0 Discovery Response Data Format 53
Table 11: Namespace Level 0 Discovery Header Format 53
Table 12: Namespace Level 0 Discovery Feature Codes 54
Table 13: Level 0 Discovery – Namespace Key Per I/O Capabilities Feature Descriptor 54
Table 14: TPER_RESET Command 56
Table 15: Clear Single MEK Command Request 58
Table 16: Clear Single MEK Command Response Payload 58
Table 17: Clear Single MEK Pending 59
Table 18: Clear All MEKs Command Request 60
Table 19: Clear All MEKs Command Response Payload 60
Table 20: Clear All MEKs Pending 61
Table 21: Get Nonce Command Block 62
Table 22: Get Nonce Response Payload 62
Table 23: ComID Assignments 67
Table 24: Supported Tokens 68
Table 25: reset_types 69
Table 26: Properties Requirements 70
Table 27: Admin SP - SPInfo Table Preconfiguration 72
Table 28: Admin SP - SPTemplates Table Preconfiguration 73
Table 29: Admin SP - Table Table Preconfiguration 73
Table 30: Admin SP - MethodID Table Preconfiguration 75
Table 31: Admin SP - AccessControl Table Preconfiguration 76
Table 32: Admin SP - ACE Table Preconfiguration 82
Table 33: Admin SP - Authority Table Preconfiguration 83

Table 34: Admin SP - C_PIN Table Preconfiguration 84
Table 35: Admin SP – TPerInfo Columns 84
Table 36: Admin SP - TPerInfo Table Preconfiguration 85
Table 37: Admin SP - Template Table Preconfiguration 85
Table 38: Admin SP - SP Table Preconfiguration 86
Table 39: data_removal_mechanism Type Table Addition 86
Table 40: data_removal_mechanism Enumeration Values 86
Table 41: DataRemovalMechanism Table Description 87
Table 42: Admin SP – DataRemovalMechanism Table Preconfiguration 87
Table 43: Key Per I/O SP - SPInfo Table Preconfiguration 88
Table 44: Key Per I/O SP - SPTemplates Table Preconfiguration 88
Table 45: Key Per I/O SP - Table Table Preconfiguration 88
Table 46: kek_obj_uidref type 90
Table 47: kek_obj_uidref_list type 90
Table 48: KeyEncryptionKey_Type 90
Table 49: Key Per I/O SP - MethodID Table Preconfiguration 90
Table 50: Key Per I/O SP - AccessControl Table Preconfiguration 91
Table 51: Key Per I/O SP - ACE Table Preconfiguration 100
Table 52: Key Per I/O SP - Authority Table Preconfiguration 102
Table 53: Key Per I/O SP - C_PIN Table Preconfiguration 103
Table 54: Key Per I/O SP - Certificates Table Preconfiguration 104
Table 55: KPIOPolicies Table Definition 104
Table 56: KPIOPolicies Table Preconfiguration 108
Table 57: KeyTagAllocation Table Definition 109
Table 58: KeyTagAllocation Table Preconfiguration 112
Table 59: KeyEncryptionKey Table Definition 112
Table 60: KeyEncryptionKey Table Preconfiguration 115
Table 61: LifeCycle Type Modification 122
Table 62: LifeCycle Type Enumeration Values 122
Table 63: Table Table Additional Column 123
Table 64: ValidMandatoryGranularity Definition 124
Table 65: ValidRecommendedGranularity Definition for Set 125
Table 66: ValidRecommendedGranularity Definition for Get 125
Table 67: Key Per I/O and KMIP Terminology Mapping 128
Table 68: Object Group Values Definition for Symmetric Key Object 133
Table 69: KMIP's Vendor Attribute's Attribute Name and Attribute Value Usage with Key Per I/O 134

Table 70: Supported LinkedObjectIdentifier Values for Symmetric Key Object 140

List of Figures

Figure 1: Example Key Per I/O's Operating Model in a Multi-Tenant Storage System 21
Figure 2: Example Diagram Legend 24
Figure 3: Key Generation with a Key Management Server Example 25
Figure 4: Plaintext KEK Provisioning Example 27
Figure 5: Importing a Storage Device's Key Per I/O Public Key Certificate into KMS and Using it for KEK Transport 29
Figure 6: Updating Key Encryption Keys Using the old Key Encryption Key 31
Figure 7: MEK Injection Example 33
Figure 8: Key Selection Example 35
Figure 9: TPer - Host Communication for Protocol ID 0x03 63
Figure 10: TCG ComPacket with KMIP Request or Response Message 66
Figure 11: Life Cycle State Diagram for Manufactured SPs 120
Figure 12: Key Per I/O KMIP Request Header 131
Figure 13: KMIP's Vendor Identification Usage for Key Per I/O 134
Figure 14: Replay Protection Example with Vendor Attribute – Adding Nonce to a Key 139
Figure 15: Replay Protection Example with Vendor Attribute - Retrieving Key Wrapped with a Nonce 139
Figure 16: Specifying KEK, Its Attributes, and Plaintext Key Data with KMIP's Batch Item 141
Figure 17: Specifying Key Data for Wrapped KEK 142
Figure 18: Specifying an MEK and its Attributes with KMIP's Batch Item 147
Figure 19: XTS-AES Key2's Link Type Attribute Linking to the KeyUID of the XTS-AES Key1 148
Figure 20: Response Message Header Structure 152
Figure 21: Response Message Batch Items 153

1 Introduction

1.1 Document Purpose

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

1.2 Scope and Intended Audience

This specification defines the Key Per I/O Security Subsystem Class (SSC). Any Storage Device that claims compliance to the Key Per I/O SSC SHALL conform to this specification.

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

1.3 Conventions

1.3.1 Key Words

Key words are used to signify requirements.

The Key Words “SHALL”, “SHALL NOT”, “SHOULD,” and “MAY” are used in this document. These words are a subset of the RFC 2119 and RFC 8174 (see [1] and [2]) key words used by TCG. These key words are to be interpreted as described in [1].

In addition to the above key words, the following are also used in this document to describe the requirements of particular features, including tables, methods, and usages thereof:

- Mandatory (M): When a feature is Mandatory, the feature SHALL be implemented. A Compliance test SHALL validate that the feature is operational.
- Optional (O): When a feature is Optional, the feature MAY be implemented. If implemented, a Compliance test SHALL validate that the feature is operational.
- Excluded (X): When a feature is Excluded, the feature SHALL NOT be implemented. A Compliance test SHALL validate that the feature is not operational.
- Not Required (N): When a feature is Not Required, the feature MAY be implemented. No Compliance test is required.

1.3.2 Font Conventions

Names of methods and 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`. KMIP protocol elements and terms are in Avenir Next LT Pro Light. All other text is in the Arial font.

1.3.3 Statement Types

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.

EXAMPLE: 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

1.3.4 SP Table Cell Color Legend

The legend in Table 1 defines the SP table cell color coding, with the RGB values for the shading of each cell indicated in parentheses. This color coding is informative only. The table cell content is normative.

If a table cell is empty or blank, regardless of shading, then the contents of that cell are not specified in this specification. The contents may be specified in another specification.

Table 1: SP Table Legend

Table Cell Legend	R-W	Value	Access Control	Comment
Arial-Narrow (230, 230, 230)	Read-only	Key Per I/O SSC specified	Fixed	• Cell content is Read-Only. • Access control is fixed. • Value is specified by the Key Per I/O SSC
Arial Narrow bold-under (230, 230, 230)	Read-only	VU	Fixed	• Cell content is Read-Only. • Access Control is fixed. • Values are Vendor Unique (VU). A minimum or maximum value may be specified.
Arial-Narrow (0, 0, 0)	Not Defined	(N)	Not Defined	• Cell content is (N). • Access control is not defined. • Any text in table cell is informative only. • A `Get` MAY omit this column from the method response.
Arial Narrow bold-under (179, 179, 179)	Write	Preconfigured, user personalizable	Preconfigured, user personalizable	• Cell content is writable. • Access control is personalizable • `Get` Access Control is not described by this color coding

Table Cell Legend	R-W	Value	Access Control	Comment
Arial-Narrow (179, 179, 179)	Write	Preconfigured, user personalizable	Fixed	• Cell content is writable. • Access control is fixed. • `Get` Access Control is not described by this color coding

1.3.5 List Conventions

1.3.5.1 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 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.

1.3.5.2 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). Each list item shall start 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 shall be indented and each item in the nested unordered list shall start 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.

1.3.5.3 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). Each listed 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 shall be indented and each item in the nested unordered list shall start 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.

1.3.6 Numbering

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 may 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 specification 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 specification 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).

1.3.7 Bit Conventions

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

1.3.8 Number Range Convention

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).

1.3.9 Specify and Provide Convention

In a given message, command, or other information exchange between a requestor and a responder:

a) the requestor specifies information in the request; and

b) the responder provides information in the response.

1.4 Document References

1.4.1 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; and
2. approved references (see section 1.4.2).

Each approved reference may specify its own document precedence.

1.4.2 Approved References

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

[2] IETF RFC 8174, 2017, “Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words’

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

[4] NIST, FIPS-197, 2001, “Advanced Encryption Standard (AES)”

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

[6] Trusted Computing Group (TCG), “TCG Storage Security Subsystem Class: Opal”, Versions 1.00, 2.00, 2.01, 2.02

[7] Trusted Computing Group (TCG), “TCG Storage Opal SSC Feature Set: PSID”, Version 1.00

[8] Trusted Computing Group (TCG), “TCG Storage Feature Set: Block SID Authentication”, Version 1.00

[9] OASIS, “OASIS Key Management Interoperability Protocol Specification”, Version 2.0

[10] OASIS, “OASIS Key Management Interoperability Protocol Usage Guide”, Version 2.0

[11] NIST, SP800-38F, 2012, “Recommendation for Block Cipher Modes of Operation: Methods for Key Wrapping”

[12] NIST, SP800-38D, 2007, “Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC”

[13] NIST, SP800-56B, 2019, “Recommendation for Pair-Wise Key-Establishment Using Integer Factorization Cryptography”

[14] NIST, FIPS 186-4, 2013, “Digital Signature Standard (DSS)”

[15] NVM Express Ratified TP 4055. Available from https://www.nvmexpress.org/

[16] NVM Express, “TCG and NVM Express Joint White Paper: TCG Storage, Opal, and NVMe”. Available from https://www.nvmexpress.org/

[17] IEEE Std 1619-2018, “IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices”, Institute of Electrical and Electronics Engineers, Inc., Jan. 25, 2019

[18] ITU-T Recommendation X.690, ISO/IEC 8825-1:2008, “Information Technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)”, November 2008

1.5 Dependencies on Other Feature Sets

This specification does not depend upon any other feature sets.

1.6 Interactions with Other Feature Sets

In the event of conflicting information in this specification and other documents, the precedence for requirements is:

1. This specification
2. TCG Storage Interface Interactions Specification
3. TCG Storage Architecture Core Specification

1.7 Definition of Terms

Term	Definition
Eradicate	irrevocably erase (e.g., cryptographically erase)
Manufactured SP	A Manufactured SP is an SP that was created and preconfigured during the Storage Device manufacturing process
N/A	Not Applicable
Original Factory State (OFS)	The original state of an SP when it was created in manufacturing, including its table data, access control settings, and life cycle state. Each Manufactured SP has its own Original Factory State. Original Factory State applies to Manufactured SPs only.
SD	Storage Device
System	A storage environment that enrolls a Key Per I/O Storage Device.
Vendor Unique (VU)	These values are unique to each Storage Device manufacturer. Typically, VU is used in table cells.
Namespace Key Tag	A Key Tag associated with a specific Namespace.
Namespace managed by the Key Per I/O SP	A Namespace that is enabled for Key Per I/O functionality for the purpose of data at rest encryption. When a Namespace is managed by the Key Per I/O SP, all interface read commands and write commands must be accompanied with a Key Tag that identifies which encryption key to use for encrypting or decrypting user data as part of the command. If a Namespace is not managed by the Key Per I/O SP, then any encryption for the Namespace will be controlled by the Storage Device without any interactions with the Key Per I/O SP.
Key Tag	A Key Tag identifies a specific Key Tag Slot that is associated with a specific Namespace. The scope of a Key Tag value is the associated Namespace.
Key Tag Slot	A Key Tag Slot is a specific location in a TPer's Media Encryption Key cache. Key Tag Slots are identified using a Key Tag for a Namespace managed by the Key Per I/O SP. An MEK stored in a Key Tag Slot can be overwritten using the Inject MEK operation

Term	Definition
	or cleared from the Key Tag Slot using the Clear Single MEK or Clear All MEKs commands.
Key Encryption Key (KEK)	A key used to encrypt another key using a key encryption/wrapping algorithm.
KPIO	Key Per I/O
Media Encryption Key (MEK)	A key intended to be used for when encrypting user data to be stored in non-volatile media.
Nonce	A random value intended to be used once per key injection request message to make successful replay attacks difficult to achieve.
Namespace association to specific `KeyTagAllocation` Table row	This specification uses the shorthand phrase “the Namespace associated with a specific `KeyTagAllocation` Table row” and other similar phrases. For the purpose of this specification, these phrases should be interpreted as follows: The Namespace that is associated with the NamespaceID that is specified in the NamespaceID column of the specific `KeyTagAllocation` Table row.
KEK association to specific `KeyTagAllocation` Table row	This specification uses the shorthand phrase “a KEK associated with a specific `KeyTagAllocation` Table row” and other similar phrases. For the purpose of this specification, these phrases should be interpreted as follows: A Key Encryption Key stored in a specific Key column of any `KeyEncryptionKey` Table row that is identified via a `KeyEncryptionKey` Table row UID specified in the AllowedKeyEncryptionKeys column value of the `KeyTagAllocation` Table row.
Preconfiguration Data	The default data in the OFS.
Pre-shared Key	A key that is established between a host and a Storage Device that is used to authenticate the host to the Storage Device and vice-versa.
Access Locked	Access Locked indicates that a Key Encryption Key is in a locked state and the use of that key is blocked. When a Key Encryption Key is in an Access Locked state, it cannot be updated, and it cannot be used to protect Media Encryption Keys or other Key Encryption Keys.

Term	Definition
Access Unlocked	Access Unlocked indicates that a Key Encryption Key is in an unlocked state and the use of that key is allowed. When a Key Encryption Key is in an Access Unlocked state, it can be updated inside a Storage Device, and it can be used to protect Media Encryption Keys or other Key Encryption Keys.

2 Key Per I/O SSC Overview

2.1 Key Per I/O SSC Use Cases and Threats

Start of informative comment

The Key Per I/O Security Subsystem Class (SSC) is an implementation profile that supports Storage Devices' "Data at Rest (DAR)" protection of users' data against unauthorized access. Building on the NVME's Key Per I/O feature (see [15]), Key Per I/O SSC provides security management interfaces that enable a host to control and specify Media Encryption Keys (MEKs) that a Storage Device uses for user data encryption.

Figure 1 depicts a high-level description of Key Per I/O's operating model.

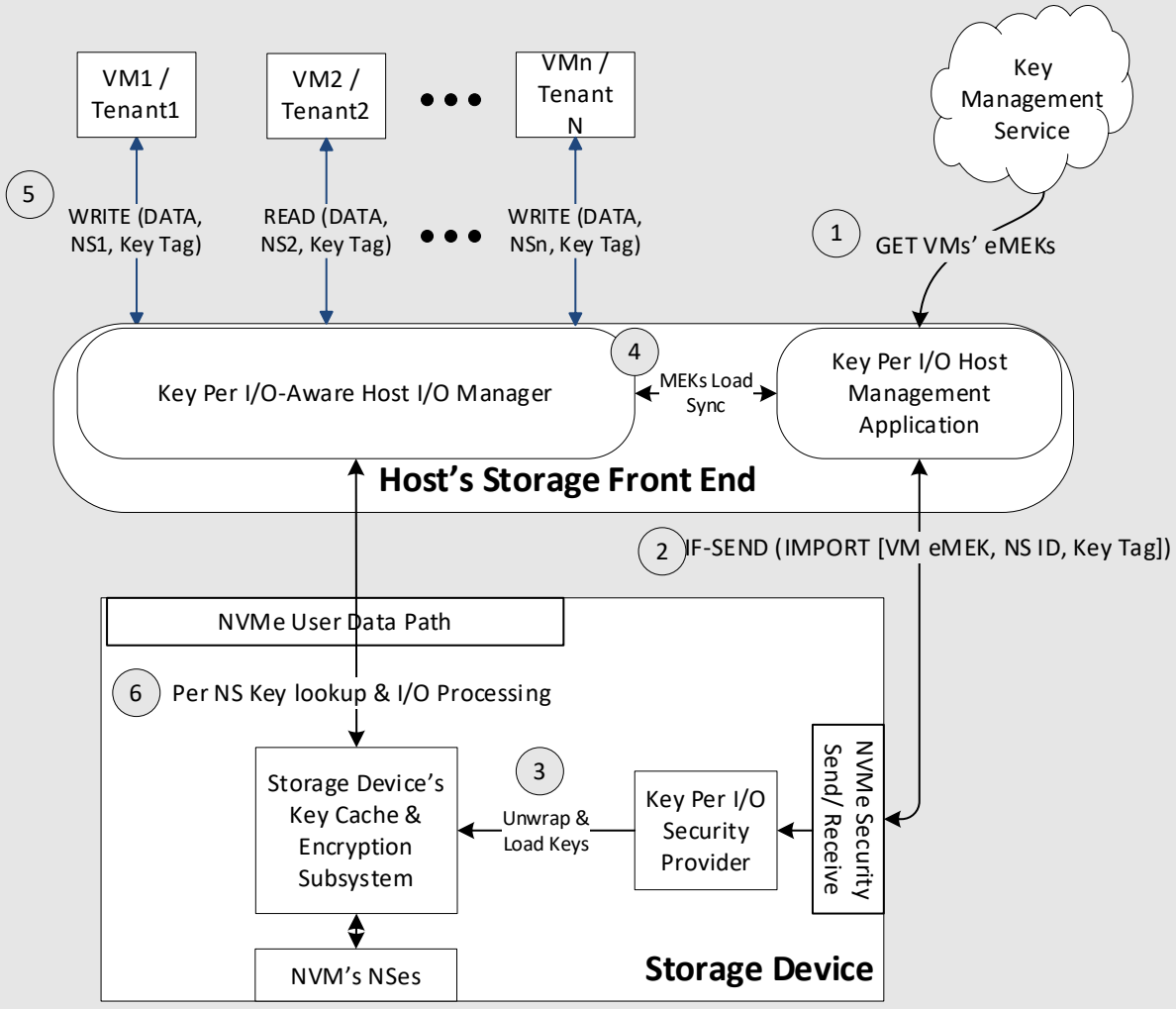


Figure 1: Example Key Per I/O's Operating Model in a Multi-Tenant Storage System

Figure 1 illustrates storage systems where a Storage Device is shared by multiple hosts and/or multiple tenants on a single host (e.g., VM/Containers). Key Per I/O provides the capability to provision Media Encryption Keys belonging to users, users' applications, or VMs' tenants hosted by those systems (for example) into specific slots in a Storage Device's volatile key cache. Subsequent read and write hosts' I/O commands can select the Media Encryption Keys to be used by that Storage Device to decrypt or encrypt the users' data.

Since the Media Encryption Keys can come from a variety of sources external to the Storage Device, data can be encrypted by keys that are known to only a particular host and/or tenant.

This is a departure from existing self-encrypting drive architectures, such as those described in [16], that perform encryption on user data based on contiguous range of LBAs using Media Encryption Keys generated and held internally in the non-volatile media of the Storage Device.

The steps shown encircled in Figure 1 are a high-level model of how Key Per I/O is used:

1. The host's storage system front-end implements a Key Per I/O management application that is responsible for managing the lifecycle of the Key Per I/O feature on a Storage Device. This responsibility includes the acquisition of Media Encryption Keys and their Key Encryption Keys (KEKs) from the host's users, and the provisioning of those keys into a Storage Device. Depending on the key management models supported by the host, the Media Encryption Keys and their Key Encryption Keys can come from an external key management service or from individual users or the users' applications management software (e.g., virtual machine monitors).
2. The host provisions the Storage Device's key cache with Media Encryption Keys, and specifies the Namespace ID and slots (i.e., Key Tags) that the Storage Device will associate with those Media Encryption Keys.
3. The Storage Device authenticates the Media Encryption Keys by unwrapping them using the host's Key Encryption Keys, and loads the Media Encryption Keys in the specified key slots after successful authentication.
4. Once the Storage Device has successfully loaded the Media Encryption Keys, it notifies the host that it is ready to accept and process the host's I/O read and write commands.
5. The host's I/O read and write commands specify the Namespace ID and the Key Tag that correspond to the media encryption key the host I/O command wants to use.
6. The Storage Device uses the Namespace ID and the Key Tag to look up the correct Media Encryption Key in its volatile key cache and uses that key for encryption or decryption of the host users' data.

By allowing a host to own, to provision, and to select which Media Encryption Keys are used by a Storage Device, Key Per I/O SSC is optimized to:

- Provide storage environments, such as those depicted in Figure 1, with fine-grain control of data encryption on the Storage Device, via Media Encryption Keys that the host can associate with a contiguous or a non-contiguous range of LBAs.
- Protect the confidentiality of stored user "Data at Rest (DAR)" against unauthorized access once it leaves the owner's control (following a power cycle and subsequent de-authentication) by ensuring the hosts' provided Media Encryption Keys are always lost on power cycles.
- Enable interoperability between Storage Devices vendors and external key management systems used for managing the injection of Media Encryption Keys into Storage Devices.
- Enable a host to encrypt its data using the same Media Encryption Key(s) at a higher abstraction level than individual drives or volumes.
- Enable efficient scaling of Media Encryption Keys usage on the Storage Device without complex key management designs as the management of the Media Encryption Keys' entire lifecycle is external to the Storage Device.
- Enable support for media sanitization when data is spread over many drives and mixed with other data that needs to be preserved.
- Enable a host to assign a Media Encryption Key to a single sensitive file or host object.

This specification addresses a limited set of "Data at Rest (DAR)" protection use cases for a Storage Device. They are:

- Deploy Storage Device & Take Ownership: A Storage Device is integrated into its target host and ownership transferred by setting or changing the Storage Device's owner credential.
- Activate or Enroll Storage Device: Authentication credentials, Key cache slots, and Key Encryption Keys (KEKs) used to protect Media Encryption Keys (MEKs) across a Storage Device's interfaces are configured. Access control is configured for KEK access.
- Lock & Unlock Storage Device: Unlocking of one or more KEKs by the host and locking of those KEKs under host control via either an explicit lock or implicit lock triggered by a reset event.
- Media Encryption Key injection: Media Encryption Keys (MEKs) are injected and associated with specific key slots that are later specified during host I/O for the purpose of encryption and decryption of all user data within a Storage Device.
- Repurpose & End-of-Life: Erasure of Key Encryption Keys (KEKs) and reset of KEKs' access credential(s) for Storage Device repurposing or decommissioning.

Therefore, a Key Per I/O SSC compliant Storage Device:

- Facilitates feature discoverability.
- Provides some user definable features (e.g., access control, key configuration, admin passwords, etc.).
- Supports Key Per I/O SSC unique behaviors (e.g., communication, table management).

End of informative comment

2.2 Provisioning Keys into a Storage Device

Start of informative comment

Key Per I/O provides a method and an interface to a host for injecting KEKs and MEKs into a Storage Device (see sections 2.2.3, 2.2.4, and 5.4.4.2 for details) and provides an interface for selecting which of those keys are used by the Storage Device during host I/O for encrypting and decrypting user data within that Storage Device (see section 2.2.5 for details).

This section describes requirements related to KEK injection, MEK injection, and MEK selection that a Key Per I/O Storage Device is expected to adhere to in order to comply with this specification.

Additionally, where possible, this section provides examples that show applications of these requirements in a storage infrastructure that enrolls Key Per I/O SSC compliant Storage Devices. In these examples, it is assumed that the entire lifecycle of keys is managed by an external entity (e.g., a Key Management Server) that is logically separate from the host entity responsible for configuring Key Per I/O SP (e.g., a Key Per I/O host management application). It is also assumed that only the Key Management Server has access to the plaintext keys before they are injected into the Storage Device.

However, some host use cases may necessitate flexible control of how, when, and what keys are loaded into a Storage Device's Namespaces, which could result in keys being generated and managed by the host's internal key management services instead of utilizing an external Key Management Server. In such cases, plaintext keys could be present in the host's system memory. Security requirements related to how the host's Key Per I/O management applications and key management services interact with each other for the purpose of exchanging and managing keys are outside the scope of this specification. However, as the security of user data depends on the security of the keys, it is strongly recommended that a Key Per I/O host management application ensures that keys are securely managed as they are transported from the entity that generates them to a Storage Device.

As subsequent sections of this specification indicate, Key Per I/O SSC allows for the transportation of host managed keys into a Storage Device to be accomplished using the Key Management Interoperability Protocol (KMIP) encoded over standard TCG synchronous communication protocols.

In this section, all example diagrams for various key provisioning models leverage KMIP operations and other KMIP constructs to help illustrate how keys are transported into a Storage Device. Readers are encouraged to familiarize themselves with KMIP specifications to understand the examples shown in this section.

For details on how exactly the KMIP protocol is encoded onto the TCG communication protocol, which of KMIP's protocol elements are required for Key Per I/O usage, and how a Key Per I/O Storage Device interprets those elements when they are invoked by the host, see sections 3.3.1, 5.4, and 5.4.4 respectively.

End of informative comment

2.2.1 Example Diagram Conventions

Start of informative comment

The example diagrams in the subsections of section 2.1 use the symbols depicted in Figure 2.

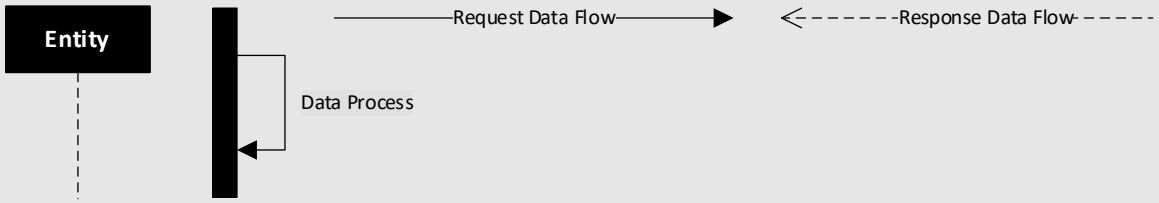


Figure 2: Example Diagram Legend

The symbols depicted in Figure 2 are described as follows:

- Entity: This symbol represents a major component that acts to generate, process, or receive data in the system
- Data Process: This symbol represents some form of process to generate, transform, or otherwise modify data in the system.
- Request Data Flow: This symbol represents the flow of a request from an entity or data process to another entity or data process. The direction of the arrow indicates the direction of the request.
- Response Data Flow: This symbol represents the flow of a response to a request from an entity or data process to another entity or data process. The direction of the arrow indicates the direction of the response.

The diagrams in the subsections of section 2.2 contain numbers for each major data flow or data process step to convey the sequence of events within the diagram. A more detailed description for each of the steps is described after each diagram.

End of informative comment

2.2.2 Key Generation

This section provides an example of key generation in a system that leverages Storage Devices that support Key Per I/O.

This example is the basis of all other examples in section 2.2.

Start of informative comment

While this example described in this section does not explicitly portray a Storage Device that supports Key Per I/O, it is useful as a base example to understand where keys and key unique identifiers exist in the system prior to any key being injected into a Storage Device that supports Key Per I/O.

Figure 3 depicts how key generation could work for a system that leverages Storage Devices that support the Key Per I/O SSC.

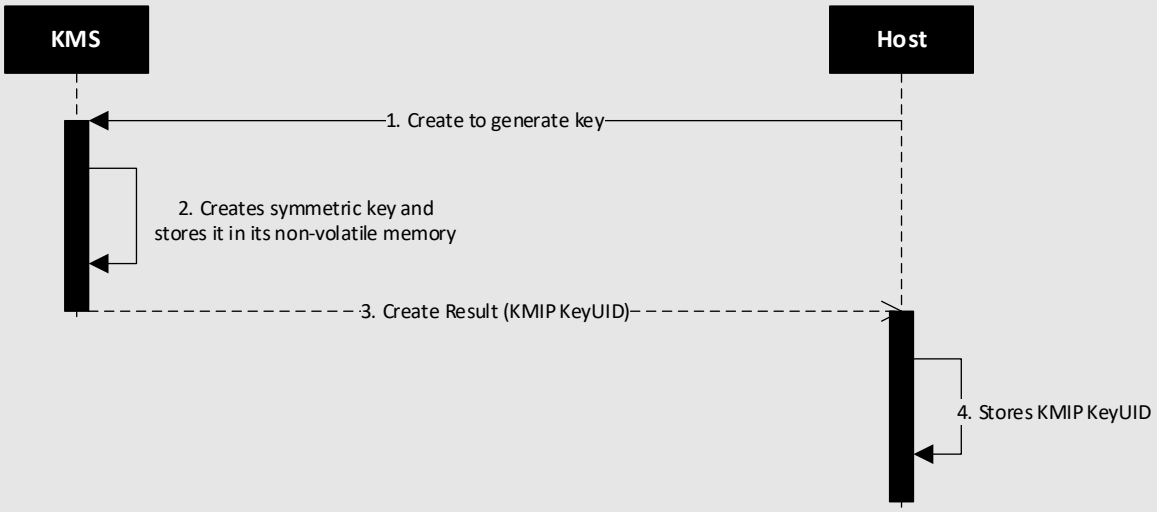


Figure 3: Key Generation with a Key Management Server Example

Figure 3 uses symbols as depicted in section 2.2.1.

Figure 3 depicts the following entities:

- Key Management Server (KMS): A server that contains all MEKs and KEKs for a given system. This server may be anywhere including both within and outside of the data center where the Host system is physically located. Requirements related to this server are outside the scope of this specification.
- Host: The system that is in communication with the KMS and is also attempting to use a Storage Device that supports Key Per I/O SSC to inject keys for later use during read operations and write operations (see sections 2.2.3, 2.2.4, and 5.4.4.2 for details). Requirements related to the Host are outside the scope of this specification.

The steps shown in Figure 3 are described as follows:

1. The Host requests the Key Management Server (KMS) to create a new key using the Create operation (see [9]) with the Object Type field set to Symmetric Key.
2. The Key Management Server (KMS) generates a new symmetric key and stores the newly generated symmetric key and associates it with a previously unused KMIP KeyUID.
3. The KMS returns the selected KMIP KeyUID to reference this key in future operations.
4. The Host stores the KMIP KeyUID and associates it with any object or entity that this key will be used with.

End of informative comment

2.2.3 KEK Provisioning Models

Start of informative comment

A host is expected to establish Key Encryption Keys (KEKs) within the Storage Device so the host can inject MEKs wrapped by those KEKs, to allow the Storage Device to recover and authenticate the MEKs. Once the MEKs have been successfully recovered and authenticated, they are loaded into the Storage Device's key cache so that they can be used during host I/O for encrypting and decrypting user data.

From a host's storage infrastructure perspective, the owner of a KEK that is used to protect an MEK represents the owner of that MEK. The owner can be a user of the storage infrastructure, a user's application, or the host's various key management services. Requirements related to the host's ownership models of the KEKs or MEKs are outside the scope of this specification.

End of informative comment

A Key Per I/O Storage Device SHALL support at least one of the following key encryption key (KEK) provisioning models in order to enable protection and authentication of media encryption keys (MEKs) as they are injected into a Storage Device:

1. Static Pre-Shared Plaintext Key Encryption Keys
2. PKI-based Key Encryption Keys Transport

2.2.3.1 Static Pre-shared Plaintext Key Encryption Keys

In this model, after a successful Key Per I/O SP activation, a host transports the initial KEKs into a Storage Device in plaintext over a secure protocol (such as PCIe IDE or SPDM Secure Sessions) or in a secure manufacturing environment. For subsequent KEK updates, the new KEKs are wrapped using authenticated encryption using KEKs that were initially provisioned over a secure medium or in a secure manufacturing process, and then injected into a Storage Device.

A Key Per I/O compliant Storage Device SHALL retain KEKs provisioned using this model across power cycles.

2.2.3.1.1 Provisioning Plaintext Key Encryption Keys

A Storage Device that supports Key Per I/O SSC MAY support the ability to allow the host to provision initial KEKs in plaintext form using appropriate commands as specified by section 5.4.4.2.3. Once this option is enabled by the host (see section 4.3.5.1.5), a Storage Device SHALL accept, process, and store persistently the injected KEKs in the appropriate `KeyEncryptionKey` Table rows as specified by the host in the key injection request.

A Storage Device that supports the provisioning of plaintext KEKs:

a) SHALL set the Plaintext KEK Provisioning Supported bit field of the Level 0 data structure to one so the host can discover that this option is supported (see section 3.1.1.3.23);
b) SHALL support the PlaintextKEKProgrammingEnabled policy as defined in section 4.3.5.1.5 to allow the enablement of this option by the host; and
c) SHALL support a `KeyEncryptionKey` Table configuration that allows the host to selectively enable or disable plaintext KEK provisioning for a single `KeyEncryptionKey` Table row (see section 4.3.5.3.10 for details).

It is strongly recommended that initial KEK provisioning occurs either in a trusted environment (e.g., a secure manufacturing process) or using a secure protocol (e.g., PCIe IDE, SPDM Secure Sessions) in an untrusted environment to ensure the security of subsequent KEK updates that use these initial keys.

Start of informative comment

Figure 4 below is an example of how an initial plaintext KEK can be provisioned into a Storage Device using a KMIP Import operation.

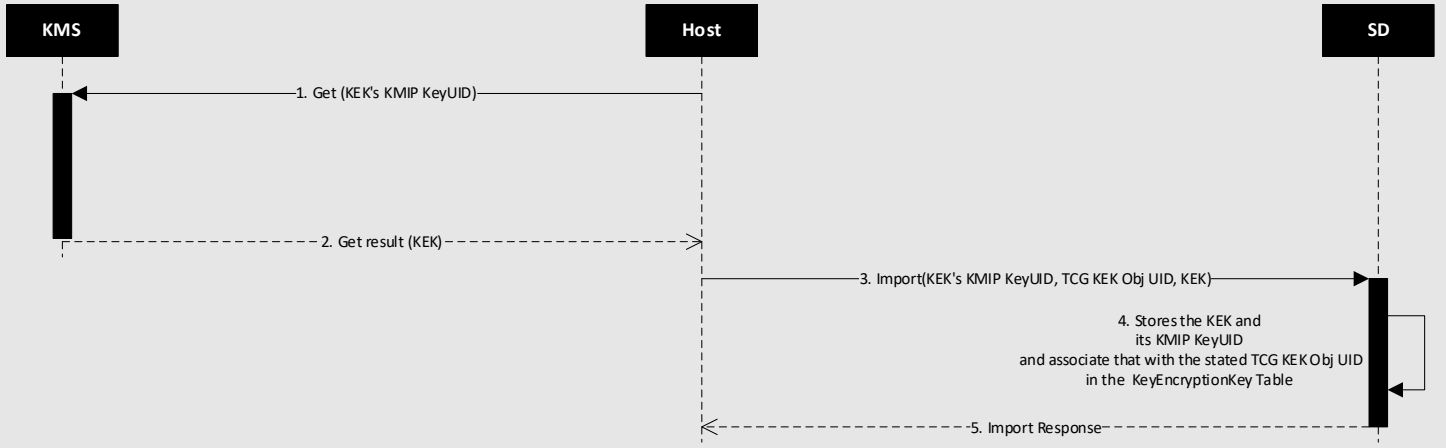


Figure 4: Plaintext KEK Provisioning Example

Figure 4 uses symbols depicted in section 2.2.1.

Figure 4 depicts the following entities:

- Key Management Server (KMS): A server that contains all MEKs and KEKs for a given system. This server may be anywhere including both within and outside the local data center where the Host system is physically located. Requirements related to this server are outside the scope of this specification.
- Host: The system that is in communication with the KMS and is also attempting to use a Storage Device that supports Key Per I/O SSC to inject keys for later use during read operations and write operations (see sections 2.2.3, 2.2.4, and 5.4.4.2 for details). Requirements related to the Host are outside the scope of this specification.
- Storage Device (SD): A Storage Device that supports Key Per I/O SSC.

The steps depicted in Figure 4 are described as follows:

1. The Host retrieves a specific KEK's KMIP KeyUID from the Host's KeyUID Store for the KEK it wants to provision within the Storage Device (SD). Note: The KEK's KMIP KeyUID was populated in the Host's KeyUID Store as described in section 2.2.2. Then, the Host requests the key associated with KEK's KMIP KeyUID from the Key Management Server (KMS) using the KMIP Get operation.
2. The Key Management Server (KMS) retrieves the Key associated with KEK's KMIP KeyUID from the KMS's Non-Volatile Key Store and returns it to the Host.
3. The Host sends a request to perform the Import operation to the Storage Device, providing the KEK, KEK's KMIP KeyUID, and TCG KEK Obj UID (e.g., the TCG UID for the object in the TCG `KeyEncryptionKey` Table that the KEK should be associated with).
4. The Storage Device stores the KEK and the KEK's KMIP KeyUID in the row associated with the stated TCG KEK Obj UID in the `KeyEncryptionKey` Table.
5. The Storage Device returns the Import result to the Host.

End of informative comment

2.2.3.2 PKI-based Key Encryption Keys Transport Mechanism

Start of informative comment

PKI-based KEKs transport helps provide confidentiality of the keys when sharing initial KEKs with a Storage Device. For subsequent KEK updates, however, it is recommended that they are performed using authenticated key wrapping

as section 2.2.3.3 describes, to enable the Storage Device to determine that the KEK being injected has not been compromised and that it originates from the same entity that provisioned the initial KEKs.

End of informative comment

A Storage Device that supports Key Per I/O MAY support the ability to allow the host to provision KEKs protected using the Storage Device's pre-provisioned Key Per I/O Public Key Certificate. Once this option is enabled by the host, the Storage Device SHALL accept, process, and store persistently the injected KEKs in the appropriate `KeyEncryptionKey` Table rows as specified by the host in the key injection request.

A Storage Device that supports PKI-based Key Encryption Key Transport:

a) SHALL set the PKI-based Key Encryption Keys Transport Supported bit field of the Level 0 data structure to one so the host can discover that this option is supported (see section 3.1.1.3.24);
b) SHALL support the PKIProtectedKEKProgrammingEnabled policy as defined in section 4.3.5.1.6 to allow the enablement of this option by the host; and
c) SHALL support a `KeyEncryptionKey` Table configuration that allows the host to selectively enable or disable PKI-based KEK provisioning for a single `KeyEncryptionKey` Table row (see section 4.3.5.3.10).

In this model, it is assumed that a Storage Device has been provisioned with an X.509 public key certificate prior to the activation of Key Per I/O SP. The public key from the certificate is used by the host for encrypting KEKs and provisioning the Storage Device. In addition, it is also assumed that the entity that needs to establish confidentiality of the KEKs with the Storage Device can successfully validate this pre-provisioned Storage Device's Key Per I/O Public Key Certificate during the enrollment of the Storage Device into the storage system.

The mechanism for obtaining, provisioning, verifying, or revoking a Storage Device's Key Per I/O Public Key Certificate is outside the scope of this specification.

2.2.3.2.1 KEK Provisioning Using Key Transport scheme such as KTS-OAEP

Start of informative comment

Figure 5 is an example showing how a Key Per I/O host management application imports a Storage Device's Key Per I/O Public Key Certificate into the KMS so the latter can use it for encrypting KEKs. In this example, it is assumed that the host's Key Per I/O management application wishes to establish confidential KEK transport between the entity that generates the KEK (in this case, a Key Management Server) and the Storage Device. As a result, the Key Management Server needs to ensure the validity of the Storage Device's Key Per I/O Public Key Certificate to

establish trust with the Storage Device and to ensure that only the intended Storage Device will be able to correctly recover the transported key material.

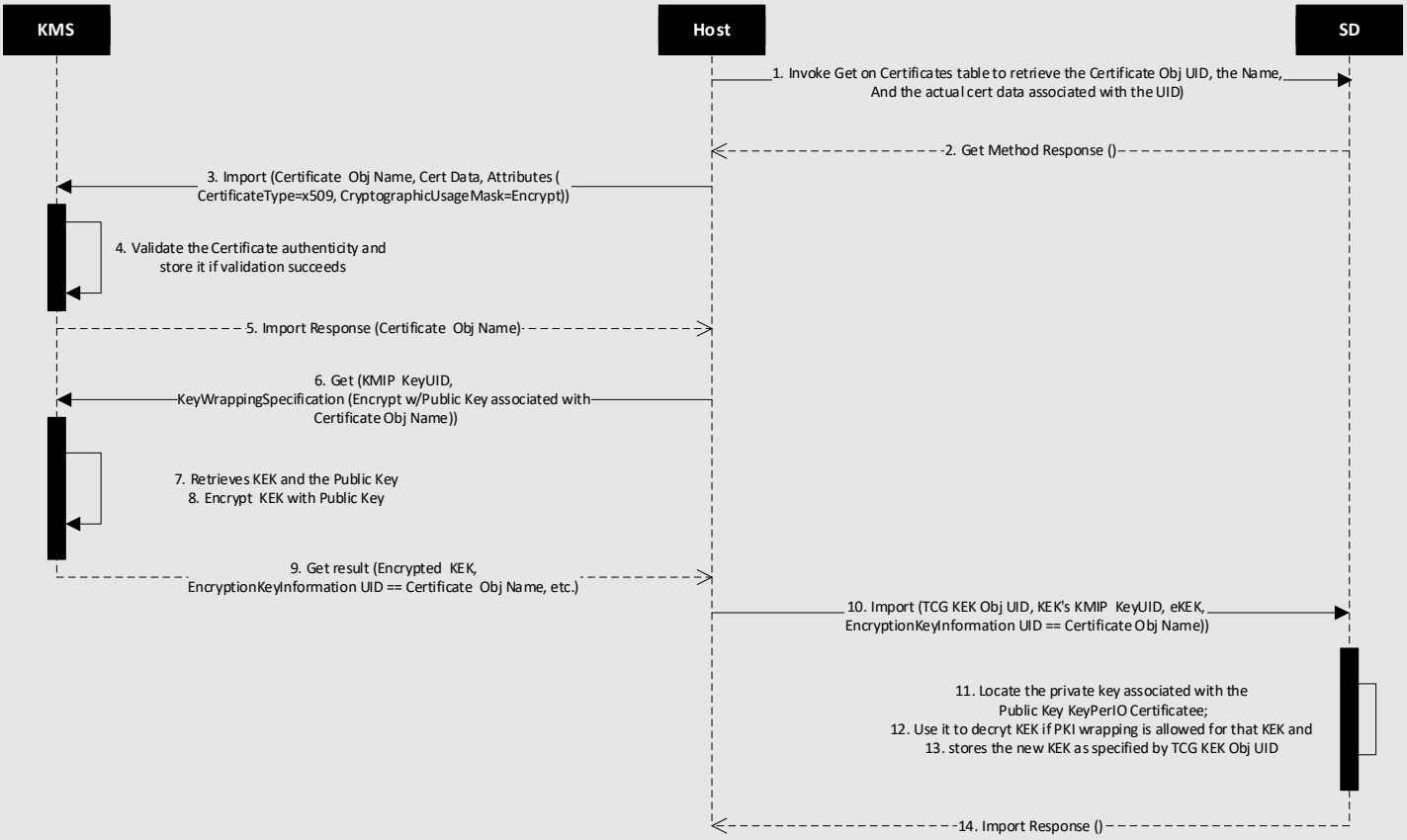


Figure 5: Importing a Storage Device's Key Per I/O Public Key Certificate into KMS and Using it for KEK Transport

Figure 5 uses symbols depicted in section 2.2.1

Figure 5 depicts the following entities:

- Key Management Server (KMS): A server that contains all MEKs and KEKs for a given system. This server may be anywhere including both within and outside the local data center where the Host system is physically located. Requirements related to this server are outside the scope of this specification.
- Host: The system that is in communication with the KMS and is also attempting to use a Storage Device that supports Key Per I/O SSC to inject keys for later use during read operations and write operations (see sections 2.2.3, 2.2.4, and 5.4.4.2 for details). Requirements related to the Host are outside the scope of this specification.
- Storage Device (SD): A Storage Device that supports Key Per I/O SSC.

The steps shown in Figure 5 are described as follows:

1. The Host retrieves the pre-provisioned Key Per I/O Public Key Certificate from the Storage Device's `Certificates` Table using the `Get` method.
2. The Storage Device returns the certificate data to the Host.

3. Then, the Host imports the certificate data into the KMS (i.e., the entity that is managing the KEKs) with the KMIP's Certificate Object's Unique Identifier value field set to the name of the Certificate Obj (i.e., the name associated with Certificate1 row in Table 54).

   When specifying the Unique Identifier of the certificate, care must be taken if the Host is managing multiple Key Per I/O Storage Devices. Since the name of the certificate object is a static value across all Key Per I/O Storage Devices, the Host should not directly use this name as the Certificate Object's Unique Identifier to the KMS. Instead, the Host should perform a Unique Identifiers translation scheme between the KMS and the individual Key Per I/O Storage Devices, then use the translated Unique Identifier in the Import operation to the KMS to ensure that each Storage Device's Key Per I/O Public Key Certificate is uniquely specified to the KMS and independently tracked within the KMS.
4. The KMS verifies the certificate's signature and, if it passes, stores the certificate data and its name (or its translated Unique Identifier) in its non-volatile memory.
5. The KMS returns the same TCG Obj Certificate1 name (or translated Unique Identifier) associated with the imported certificate to the Host.
6. The Host is now able to directly specify a Unique Identifier associated with the certificate as the Unique Identifier for the Public Key encryption key (i.e., in the Unique Identifier field of the Key Wrapping Specification's Encryption Key Information structure) for the KEK that it wants to retrieve encrypted from the KMS and transport into the Storage Device.
7. The Key Management Server (KMS) retrieves the KEK using the KEK's KMIP KeyUID and retrieves the certificate.
8. The KMS encrypts the KEK using the Key Per I/O Public Key contained in the certificate.
9. The KMS returns the Encrypted KEK and its Key Wrapping Data to the Host. The Key Wrapping Data contains information that tells the Storage Device how to decrypt the KEK by providing the name of the certificate object that contains the public key used to wrap the KEK, the encryption algorithm used, etc.
10. The Host sends a request to perform the Import operation to the Storage Device, providing the following information: TCG KEK Object UID, KEK's KMIP KeyUID, Encrypted KEK, and Key Wrapping Data that specifies how the key was wrapped and the key used to wrap the injected KEK.

    If the Host has implemented a translation of Unique Identifiers for uniquely tracking each Storage Device's Key Per I/O Public Key certificate, the Host should ensure that Import's operation Key Wrapping Data section specifies the correct Unique Identifier associated with the certificate object (i.e., the name of the Certificate1 object) as the Unique Identifier of the key used to encrypt the injected KEK.
11. The Storage Device retrieves the Private Key associated with its Key Per I/O Public Key contained in the certificate.
12. The Storage Device uses the retrieved Private Key to decrypt the encrypted KEK assuming PKI-protected KEK programming is allowed (see section 4.3.5.1.6) OR the PKIPublicKeyEncryptionKey UID in the `KeyEncryptionKey` Table is listed as one of the KEK objects allowed for the specified TCG KEK Object UID (see section 4.3.5.3.10 on how `KeyEncryptionKey` Table UIDs are used to configure a desired KEK protection scheme).
13. Then, the Storage Device stores the KEK and its KMIP KeyUID in the `KeyEncryptionKey` Table row associated with the stated TCG KEK Object UID.
14. The Storage Device completes processing the Import operation and returns the result to the Host.

At this point, the Host has successfully established a KEK in the Storage Device and can use it for protecting subsequent KEK updates or MEKs, using supported authenticated encryption algorithms if it wishes.

Using the Storage Device's Key Per I/O Public Key Certificate directly for encrypting transported key material has its limitations. For example, it does not offer cryptographic assurance that the key being injected originates from the intended platform user (i.e., there is no mutual authentication). To help minimize the risk of unauthorized parties injecting keys, the Host could lock the key injection interface when not in use, as described in section 4.3.5.1.7

Additionally, the Host could also establish an end-to-end transport link security with mutual authentication capabilities between the entity managing KEKs and the Storage Device using secure transport protocols such as PCIe IDE or SPDM Secure Sessions. The specifics of how those transport security protocols could interact with Key Per I/O are outside the scope for this specification.

End of informative comment

2.2.3.3 Updating Key Encryption Keys (KEKs) Using a Previously Provisioned KEK

A Storage Device that supports Key Per I/O SSC SHALL support updating stored KEKs using at least one of the supported authenticated key wrapping algorithms (the details of wrapping algorithms are described in section 3.1.1.3). The Storage Device SHALL accept, process, and store persistently the injected KEKs in the appropriate `KeyEncryptionKey` Table rows (see section 4.3.5.3 for details) as specified by the host in the key injection request.

Start of informative comment

Figure 6 is an example showing how a KeyEncryptionKey object can be updated with a new KEK that is protected using a different KEK that has already been provisioned in the Storage Device.

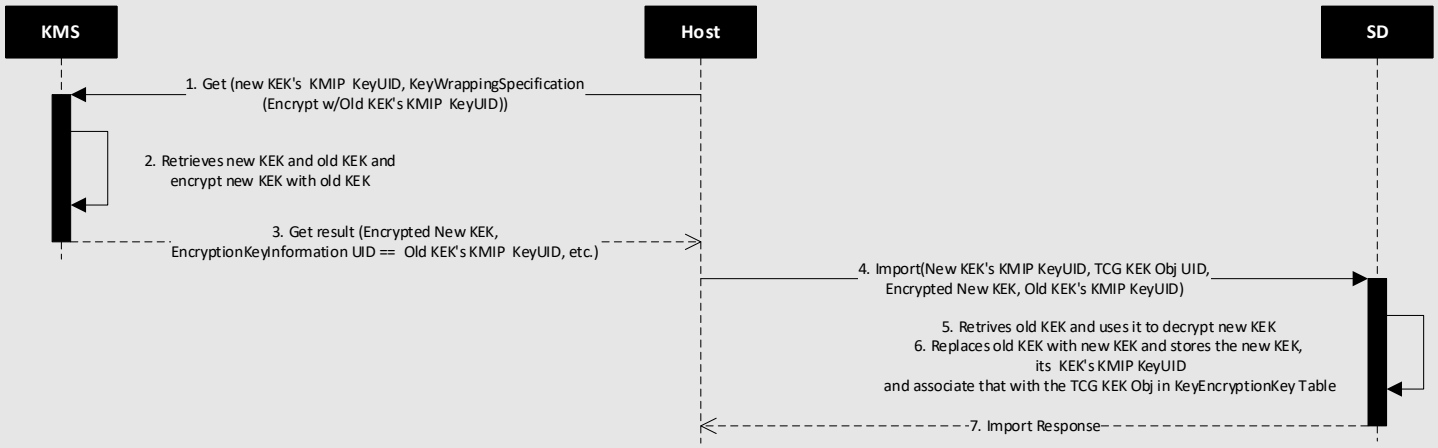


Figure 6: Updating Key Encryption Keys Using the old Key Encryption Key

Figure 6 uses symbols as depicted in section 2.2.1.

Figure 6 depicts the following entities:

- Key Management Server (KMS): A server that contains all MEKs and KEKs for a given system. This server may be anywhere including both within and outside the local data center where the Host system is physically located. Requirements related to this server are outside the scope of this specification.

- Host: The system that is in communication with the KMS and is also attempting to use a Storage Device (that supports Key Per I/O SSC) to inject keys for later use during read operations and write operations (see sections 2.2.3, 2.2.4, and 5.4.4.2 for details). Requirements related to the Host are outside the scope of this specification.
- Storage Device (SD): A Storage Device that supports Key Per I/O SSC.

The steps shown in Figure 6 are described as follows:

1. The Host retrieves both the new and old KEKs' KMIP UIDs from the Host's KeyUID Store. (Note: The new and old KEKs' KMIP UIDs were populated in the Host's KeyUID Store as described in section 2.2.2).Then, the Host requests the key associated with New KEK's KMIP KeyUID to be encrypted using the key associated with old KEK's KMIP KeyUID and returned by the Key Management Server (KMS) via the KMIP Get operation.

   If the Host has enabled Replay Protection on the Storage Device, the Host is expected to include the Nonce that it retrieves from the Storage Device in the Key Wrapping Specification structure as an attribute to be wrapped with the key material to ensure that the key material is replay-protected when it is transported to the Storage Device (see section 3.2.6 for details of the Replay Protection Feature).
2. The Key Management Server retrieves the keys associated with new KEK's KMIP KeyUID and old KEK's KMIP KeyUID from the KMS's Non-Volatile Key Store.
3. The KMS encrypts the new KEK (together with the associated Nonce attribute specified by the Key Wrapping Specification structure if the Host has enabled replay protection on the Storage Device) using the old KEK and returns the encrypted new KEK and wrapping data, which includes the old KEK's KMIP KeyUID as the KeyUID for the EncryptionKeyInformation Unique Identifier field, to the Host.
4. The Host sends a request to perform the Import operation to the Storage Device, providing the Encrypted New KEK, New KEK's KMIP KeyUID, and TCG KEK Obj UID, along with Key Wrapping Data that indicates that the new KEK was encrypted using the old KEK.
5. The Storage Device Controller (SDC) retrieves the key associated with old KEK KeyUID, if it's allowed to be used to unwrap KEKs intended for the specified TCG KEK Obj UID, and uses it to decrypt the encrypted new KEK.
6. The Storage Device replaces the old KEK with the new KEK and stores it with its KEK''s KMIP KeyUID in the `KeyEncryptionKey` Table row associated with the stated TCG KEK Obj UID.
7. The Storage Device returns the Import result to the Host.

End of informative comment

2.2.4 Media Encryption Key (MEK) Injection

A Storage Device that supports a Key Per I/O SSC SHALL support injection of Media Encryption Keys (MEKs) protected with previously provisioned KEKs using at least one of the supported authenticated key wrapping algorithms (see section 2.5 for details).

A Key Per I/O compliant Storage Device SHALL NOT store injected Media Encryption Keys in its non-volatile media.

If the Key Per I/O SP is owned (see section 5.2.2.2.1), all Media Encryption Keys injected via the KMIP interface SHALL be cleared from volatile storage media as a result of a TCG Power Cycle Reset.

Other TCG resets SHALL NOT cause Media Encryption Keys to be cleared from volatile storage media.

Start of informative comment

Figure 7 is an example to help explain how an MEK protected using a previously provisioned KEK is injected into a Storage Device that supports a Key Per I/O SSC.

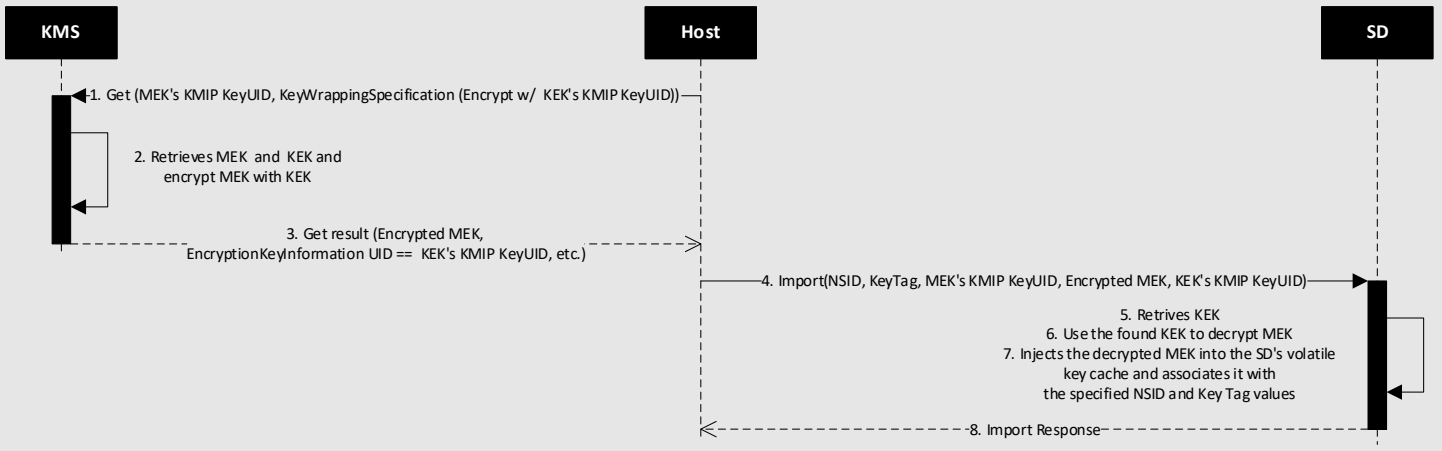


Figure 7: MEK Injection Example

Figure 7 uses symbols depicted in section 2.2.1.

Figure 7 depicts the following entities:

- Key Management Server (KMS): A server that contains all MEKs and KEKs for a given system. This server may be anywhere including both within and outside the data center where the Host system is physically located. Requirements related to this server are outside the scope of this specification.
- Host: The system that is in communication with the KMS and is also attempting to use a Storage Device (that supports Key Per I/O SSC) to inject keys for later use during read operations and write operations (see sections 2.2.3, 2.2.4, and 5.4.4.2 for details). Requirements related to the Host are outside the scope of this specification.
- Storage Device (SD): A Storage Device that supports the Key Per I/O SSC.

The steps shown in Figure 7 are described as follows:

1. The host retrieves the MEK's KMIP KeyUID and the KEK's KMIP KeyUID from the Host's KeyUID Store (Note: The MEK's KMIP KeyUID and KEK's KMIP KeyUID were populated in the Host's KeyUID store as described in section 2.2.2). Then, the host requests that the key associated with MEK's KMIP KeyUID be encrypted using the key associated with KEK's KMIP KeyUID from the Key Management Server (KMS) via a KMIP Get operation. If the host has enabled Replay Protection on the Storage Device, it should include the Nonce that it retrieves from the Storage Device in the Key Wrapping Specification structure as an attribute to be wrapped with the key material, to ensure that the key material is replay-protected when it is transported to the Storage Device (see section 3.2.6 for details of the Replay Protection Feature).
2. The KMS retrieves the MEK and KEK using the MEK's KMIP KeyUID and KEK's KMIP KeyUID respectively.
3. The KMS encrypts the MEK (together with the associated Nonce attribute specified by the Key Wrapping Specification if the host has enabled replay protection on the Storage Device) using the KEK and returns the Encrypted MEK and Key Wrapping Data to the Host. The Key Wrapping Data contains the information that helps the Storage Device decrypt the MEK, such as the KEK's KMIP KeyUID used in the EncryptionKeyInformation Unique Identifier field, the encryption algorithm used, etc.
4. The Host sends the Import operation request to the Storage Device, providing the following information: Namespace ID, Key Tag, MEK's KMIP KeyUID to be associated with the injected MEK, encrypted MEK, and Wrapping Data (Algorithm, KEK KeyUID).

5. The Storage Device uses the KEK's KMIP KeyUID specified in the Key Wrapping Data to lookup the KEK associated with the KMIP KeyUID in the KEK Store. If there exists no KEK in the KEK Store associated with KEK's KMIP KeyUID, then the Storage Device returns an error to the Host.
6. The Storage Device decrypts the encrypted MEK using the KEK, and the Encryption Algorithm specified in the Key Wrapping Data. If the decryption operation fails for some reason (for example, if the MEK is encrypted using a different KEK or KEK that isn't allowed for that Namespace (see section 4.3.5.2.7)), then the Storage Device returns an error to the Host (see section 5.4.4.2.4 for error handling related to MEK injections).
7. The Storage Device injects the decrypted MEK into its Volatile MEK Cache and associates it with the provided Namespace ID and Key Tag values to be used by subsequent I/O.
8. The Storage Device returns the result of the Import operation to the Host.

End of informative comment

2.2.5 Key Selection Theory of Operation

A Key Per I/O SSC compliant Storage Device may support multiple Namespaces and multiple Key Tags per supported NVMe Namespace.

A Key Per I/O SSC compliant Storage Device SHALL support an independent Media Encryption Key (MEK) per Namespace and Key Tag pair. The number of Namespace and Key Tag pairs is identified in the Total Number of Key Tags Supported field in the Key Per I/O SSC v1.00 feature descriptor.

The Key Tag field is a zero-based index into a virtual key cache within a specific Namespace. If a Key Per I/O SSC compliant Storage Device supports N key tags per Namespace (as defined by the Max Number of Key Tags Supported Per Namespace field in the Key Per I/O SSC Feature Descriptor), then the Storage Device SHALL support key tags from 0 to N-1 for each Namespace managed by the Key Per I/O SP.

Start of informative comment

The scope of uniqueness for a Key Tag is only at the NVMe Namespace level. Because of this, both the Namespace ID and the Key Tag values are needed to properly select a specific key slot when inserting a Media Encryption Key using an inject MEK command (i.e., KMIP Import) and to select a specific key to be used for any I/O command.

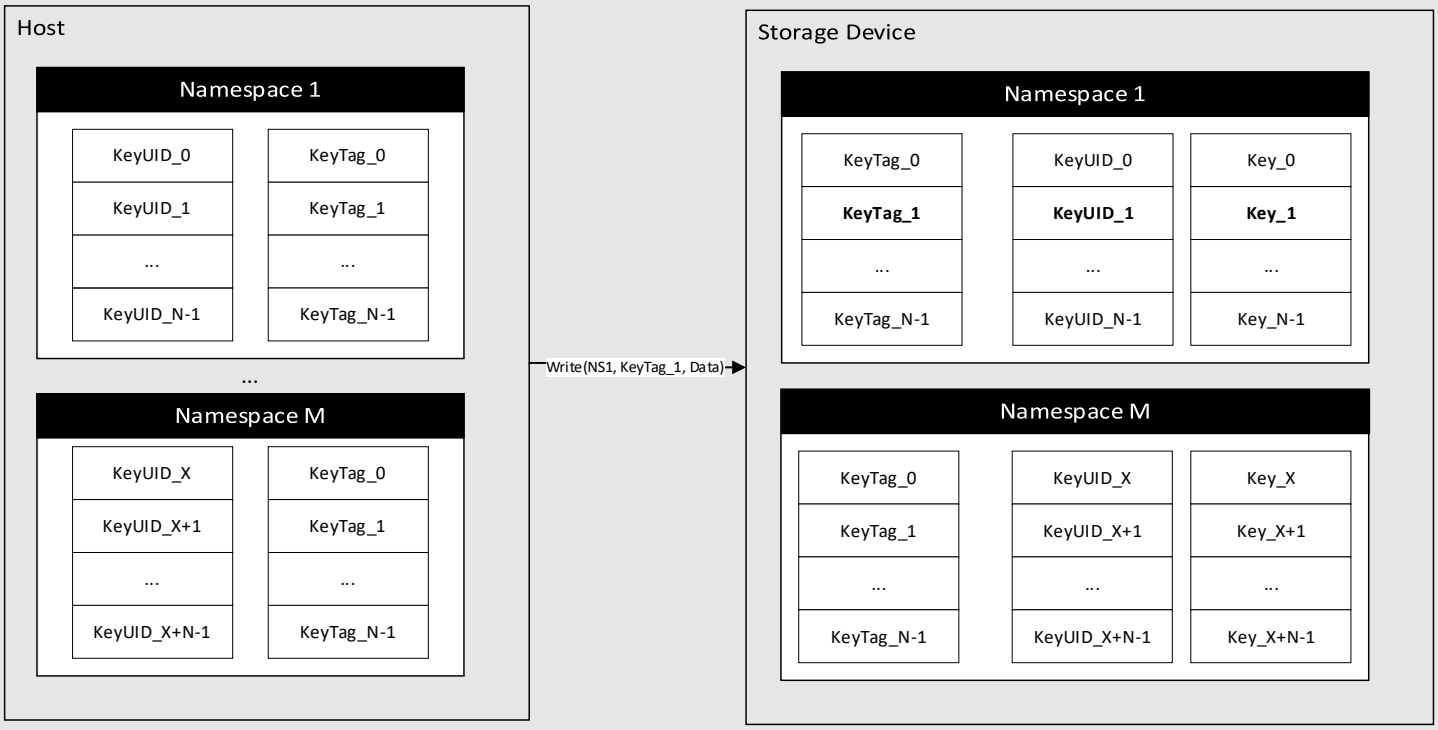


Figure 8: Key Selection Example

The following sequence of steps and Figure 8 are an example to help explain how key selection works with Key Per I/O SSC. Note that the example uses a single KeyUID to refer to an MEK but, actually, since the MEK is generally composed of two keys (i.e., XTS-AES Key1 and XTS-AES Key2), there are two keys' UIDs for each MEK. So, the mapping illustrated in Figure 8 is simply a highly abstracted version of what it looks like to track an MEK mapping to a particular NS.

For Writes:

1. The prerequisite is that the host has already injected all Media Encryption Keys it plans to use with the Storage Device using a KMIP Import operation.
2. The host selects the Namespace and the KeyUID for a specific encryption key that will be used to encrypt user data when performing a write operation with the Storage Device. In this example, the Host selects Namespace 1 and KeyUID_1 to represent the key to encrypt the data with.
3. The Host looks up the Key Tag associated with Namespace 1 and KeyUID_1. In this example, Namespace 1 and KeyUID_1 map to KeyTag_1.
4. The Host issues a Write command to the Storage Device, specifying Namespace 1 and KeyTag_1.
5. The Storage Device looks up the key associated with the specified Namespace and Key Tag. In this example, Key_1 is selected because it is currently associated with the Namespace 1 and KeyTag_1 pair.
6. The Storage Device then uses Key_1 to encrypt the data transferred along with the Write command.

7. Optionally, if the Storage Device supports Incorrect Key Detection, the Storage Device may also write some metadata (such as KeyUID_1) along with the encrypted data to non-volatile storage. This metadata would then be able to be used when processing a read operation to help the Storage Device identify whether an incorrect key is being used. Note that in this case, the Storage Device cannot determine whether the wrong key is used for the original write operation or for the read operation, just that a different key was used when the data was originally written compared to when the data is being read.
8. The Storage Device stores the encrypted data to non-volatile storage.

For Reads:

1. The prerequisite is that the host has already injected all Media Encryption Keys it plans to use with the Storage Device using the KMIP Import operation.
2. The host selects the Namespace and the KeyUID for a specific encryption key that will be used to decrypt user data when performing a read operation with the Storage Device. In this example, the Host selects Namespace 1 and KeyUID_1 to represent the key to decrypt the data.
3. The Host looks up the Key Tag associated with Namespace 1 and KeyUID_1. In this example, Namespace 1 and KeyUID_1 map to KeyTag_1.
4. The Host issues a Read command to the Storage Device, specifying Namespace 1 and KeyTag_1.
5. The Storage Device looks up the key associated with the specified Namespace and Key Tag. In this example, Key_1 is selected because it is currently associated with Namespace 1 and KeyTag_1.
6. The Storage Device then uses Key_1 to decrypt the data to be transferred to the host as a result of the Read command.
7. Optionally, if the Storage Device supports Incorrect Key Detection, the Storage Device may also use metadata stored when the data was written to help detect whether the Storage Device is using a different key during the read operation than the key that was used during the previous write operation to the same LBA range. The Storage Device can do this by comparing the metadata that is stored in non-volatile media to the metadata that would be used for the current key and/or KeyUID (Key_1 and KeyUID_1 in this example). If the Storage Device detects that an incorrect key is being used for the read command, then the Storage Device will return an error indicating that an incorrect key was used. If the Storage Device does not support Incorrect Key Detection, the behavior is vendor implementation specific.
8. If no errors have occurred during the read operation, the Storage Device will return the data decrypted using Key_1 to the host.

Note: If the Host were instead to select Namespace M and KeyTag_1, the Storage Device would instead lookup Key_X+1 and KeyUID_X+1 rather than Key_1. This is because the scope of the Key Tag field is within a specific Namespace.

End of informative comment

2.3 Security Providers (SPs)

A Key Per I/O SSC compliant Storage Device SHALL support at least two Security Providers (SPs):

1) Admin SP
2) Key Per I/O SP

2.4 Interface Communication Protocol

A Key Per I/O SSC compliant Storage Device SHALL implement the synchronous communications protocol as defined in Section 3.4.4.

2.5 Cryptographic Features

A Key Per I/O SSC compliant Storage Device SHALL implement Full Disk Encryption for all host accessible user data stored on media within the Namespaces managed by the Key Per I/O SP (see section 4.3.5.2.5 for more details on the Namespaces managed by Key Per I/O SP).

XTS-AES-256 SHALL be supported (see [4]). All other algorithms, block cipher modes, and key lengths SHALL NOT be supported for encrypting user data stored on the Storage Device.

If a TPer receives a request to inject the media encryption keys (i.e., XTS-AES-256 Key1 and Key2) where the specified Key Length of each key is not 256 bits, the TPer SHALL fail the request with an appropriate failure result reason (see section 5.4.4.2.4 for details of failure result reasons).

A Key Per I/O SSC compliant Storage Device SHALL support at least one of the following symmetric key wrapping algorithms for encrypting keys during their transportation into the Storage Device:

1. NIST AES-KW algorithm as defined in [11] (see section 3.1.1.3.16 for details).
2. NIST AES-GCM algorithm as defined in [12] (see section 3.1.1.3.17 for details).

In addition to a symmetric key wrapping algorithm above, a Key Per I/O SSC compliant Storage Device that supports PKI-based KEK Transport SHALL support NIST RSA-OAEP as defined in [13] (see section 3.1.1.3.18).

2.6 Authentication

A Key Per I/O SSC compliant Storage Device SHALL support password authorities and authentication.

2.7 Table Management

This specification defines the mandatory tables and mandatory/optional table rows delivered by the Storage Device manufacturer. The creation or deletion of tables or table rows after manufacturing is outside the scope of this specification.

2.8 Access Control & Personalization

Initial access control policies are preconfigured at Storage Device manufacturing time on manufacturer created SPs. A Key Per I/O SSC compliant Storage Device SHALL support personalization of Access Control Elements of the Key Per I/O SP as specified in section 4.3.1.6.

2.9 Issuance

The Key Per I/O SP may be present in the Storage Device when the Storage Device leaves the manufacturer. The issuance of SPs is outside the scope of this specification.

2.10 Key Per I/O SSC Discovery

Refer to [3] for details (see section 3.1.1).

2.11 Mandatory Feature Sets

A Key Per I/O SSC compliant Storage Device SHALL support the following TCG Storage Feature Sets:

1) PSID, Opal SSC Feature Set (refer to [7]).
2) Block SID Authentication Feature Set (refer to [8]).

3 Key Per I/O SSC Features

3.1 Security Protocol 1 Support

3.1.1 Level 0 Discovery (M)

Refer to [3] for more details.

A Key Per I/O SSC compliant Storage Device SHALL return the following Level 0 responses:

- Level 0 Discovery Header (see Table 2)
- TPer Feature Descriptor (see Table 3)
- Key Per I/O SSC Feature Descriptor (see Table 4)
- Namespace Level 0 Discovery Header (see Table 11)

Additionally, a Key Per I/O SSC compliant Storage Device MAY return the following Level 0 response:

- Supported Data Removal Mechanism Feature Descriptor (see Table 6)

3.1.1.1 Level 0 Discovery Header

Table 2: Level 0 Discovery Header

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
1				Length of Parameter Data				
2								
3								(LSB)
4	(MSB)							
5				Data structure revision				
6								
7								(LSB)
8	(MSB)							
…				Reserved				
15								(LSB)
16	(MSB)							
…				Vendor Specific				
47								(LSB)

A Key Per I/O SSC compliant Storage Device SHALL return the following field values:

- Length of parameter data = VU

- Data structure revision = 0x00000001 or
  any version that supports the defined features in this SSC
- Vendor Specific = VU

3.1.1.2 TPer Feature (Feature Code = 0x0001)

Table 3: Level 0 Discovery - TPer Feature Descriptor

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)							
1	Feature Code (0x0001)							(LSB)
2	Version				Reserved			
3	Length							
4	Reserved	ComID Mgmt Supported	Reserved	Streaming Supported	Buffer Mgmt Supported	ACK/NAK Supported	Async Supported	Sync Supported
5 - 15	Reserved							

A Key Per I/O SSC compliant Storage Device SHALL return the following field values:

- Feature Code = 0x0001
- Version = 0x1 or any version that supports the defined features in this SSC
- Length = 0x0C
- ComID Mgmt Supported = VU
- Streaming Supported = 1
- Buffer Mgmt Supported = VU
- ACK/NACK Supported = VU
- Async Supported = VU
- Sync Supported = 1

3.1.1.3 Key Per I/O SSC v1.00 Feature (Feature Code = 0x0305)

Table 4: Level 0 Discovery – Key Per I/O SSC Feature Descriptor

Bit Byte	7	6	5	4	3	2	1	0
0	(MSB)							
1	Feature Code (0x0305)							(LSB)
2	Feature Descriptor Version Number				SSC Minor Version Number			
3	Length							
4	(MSB)							
5	Protocol 0x01 Base ComID							(LSB)
6	(MSB)							
7	Protocol 0x01 Number of ComIDs							(LSB)
8	(MSB)							
9	Protocol 0x03 Base ComID							(LSB)
10	(MSB)							
11	Protocol 0x03 Number of ComIDs							(LSB)
12	Initial C_PIN_SID PIN Indicator							
13	Behavior of C_PIN_SID PIN upon TPer Revert							
14	(MSB)							
15	Number of Key Per I/O SP Admin Authorities Supported							(LSB)
16	Reserved	Reserved	Replay Protection Enabled	Replay Protection Supported	Incorrect Key Detection Supported	Shared XTS-AES Tweak Key Required	Key Per I/O Scope	Key Per I/O Enabled
17-18	Maximum Supported Key Unique Identifier Length							
19	Reserved							KMIP Formatted Key Injection Supported
20	Reserved							
21	Reserved					NIST RSA-OAEP Supported	NIST AES-GCM Supported	NIST AES-KW Supported
22	Reserved							

Bit Byte	7	6	5	4	3	2	1	0
23	Reserved							AES-256 Wrapping Key Supported
24	Reserved							
25	Reserved					RSA4K Wrapping Key Supported	RSA3K Wrapping Key Supported	RSA2K Wrapping Key Supported
26	Reserved							
27	Reserved						PKI-based KEK Transport Supported	Plaintext KEK Provisioning Supported
28	Reserved							
29-31	Reserved							
32-35	Number of Key Encryption Keys Supported							
36-39	Total Number of Key Tags Supported							
40-41	Maximum Number of Key Tags Supported Per Namespace							
42	Get Nonce Command Nonce Length							
43-47	Reserved							

3.1.1.3.1 Protocol 0x01 Base ComID

The Protocol 0x01 Base ComID field identifies the first statically allocated ComID supported by a TPer for use in the ComID field of the IF-SEND and IF-RECV commands. See [3] for more details on ComIDs.

The ComID identified in this field SHALL be bound to Security Protocol ID 0x01. See section 3.3.3 for more details on ComID binding to security protocols.

3.1.1.3.2 Protocol 0x01 Number of ComIDs

The Protocol 0x01 Number of ComIDs field is the total number of statically allocated ComIDs supported by the Device that are bound to Protocol ID 0x01. See [3] for more details on ComIDs and see section 3.3.3 for more details on ComID binding to security protocols.

3.1.1.3.3 Protocol 0x03 Base ComID

The Protocol 0x03 Base ComID field identifies the first statically allocated ComID supported by a TPer for use in the ComID field of the IF-SEND and IF-RECV commands.

The ComID identified in this field SHALL be bound to Security Protocol 0x03. See section 3.3.3 for more details on ComID binding to security protocols.

3.1.1.3.4 Protocol 0x03 Number of ComIDs

The Protocol 0x03 Number of ComIDs field is the total number of statically allocated ComIDs supported by the Device that are bound to Protocol ID 0x03. See section 3.3.3 for more details on ComID binding to security protocols.

3.1.1.3.5 Initial C_PIN_SID PIN Indicator

The Initial C_PIN_SID PIN Indicator identifies the initial PIN value for C_PIN_SID (e.g., the initial PIN for the SID authority) for a Storage Device created during manufacturing.

If the Initial C_PIN_SID PIN Indicator field is set to 0x00, the initial C_PIN_SID PIN value SHALL be equal to the C_PIN_MSID PIN value.

If the Initial C_PIN_SID PIN Indicator field is set to 0xFF, the initial C_PIN_SID PIN value is VU and SHALL NOT be equal to the C_PIN_MSID PIN value.

The Initial C_PIN_SID PIN Indicator field values of 0x02 – 0xFE are reserved by TCG.

3.1.1.3.6 Behavior of C_PIN_SID PIN upon TPer Revert

The Behavior of C_PIN_SID PIN upon TPer Revert field identifies the PIN value for C_PIN_SID (e.g., the PIN value for the SID authority) after successful invocation of `Revert` on the Admin SP's object in the `SP` table.

If Behavior of C_PIN_SID PIN upon TPer Revert field is set to 0x00, the C_PIN_SID PIN value SHALL become the value of the C_PIN_MSID PIN column after successful invocation of `Revert` on the Admin SP's object in the `SP` table.

If Behavior of C_PIN_SID PIN upon TPer Revert field is set to 0xFF, the C_PIN_SID PIN value SHALL change to a VU value after successful invocation of `Revert` on the Admin SP's object in the `SP` table and SHALL NOT be equal to the C_PIN_MSID PIN value.

The Behavior of C_PIN_SID PIN upon TPer Revert field values of 0x1 – 0xFE are reserved by TCG.

3.1.1.3.7 Number of Key Per I/O SP Admin Authorities

The Number of Key Per I/O SP Admin Authorities field is the number of Admin authorities supported within the Key Per I/O SP.

The Number of Key Per I/O SP Admin Authorities field value SHALL be set to a value of at least 1.

3.1.1.3.8 Key Per I/O Enabled

The Key Per I/O Enabled bit SHALL be set to one if the Key Per I/O SP is in any state other than Manufactured-Inactive.

3.1.1.3.9 Key Per I/O Scope

The Key Per I/O Scope bit determines whether the Key Per I/O capability applies to all Namespaces in the NVM subsystem when Key Per I/O SP is in any state other than Manufactured-Inactive.

If the Key Per I/O Scope bit is set to one, the Key Per I/O capability SHALL apply to all Namespaces in the NVM subsystem when Key Per I/O SP in any state other than Manufactured-Inactive.

If the Key Per I/O Scope bit is set to zero, then the Key Per I/O capability SHALL be independently enabled or disabled uniquely per each Namespace within the NVM subsystem when Key Per I/O SP is in any state other than Manufactured-Inactive (see section 4.3.5.2.5 for details).

The value of the Key Per I/O Scope bit SHALL equal the value of the Key Per I/O Scope (KPIOSC) bit of the Key Per I/O Capabilities field in NVMe's Identify Controller data structure (see [15]).

3.1.1.3.10 Shared XTS-AES Tweak Key Required

Start of informative comment

The term XTS-AES "Tweak Key" refers to "XTS-AES Key 2" of the XTS-AES algorithm (see [17] for details).

End of informative comment

The Shared XTS-AES Tweak Key Required bit indicates whether a TPer requires a shared XTS-AES Tweak Key across all Media Encryption Keys.

A Key Per I/O SSC compliant Storage Device MAY support requiring a shared XTS-AES Tweak Key across all Media Encryption Keys.

If the Shared XTS-AES Tweak Key Required bit is set to one, then the TPer SHALL support only a single shared XTS-AES Tweak Key for all Media Encryption Keys that is generated and managed by the Storage Device.

If the Shared XTS-AES Tweak Key Required bit is cleared to zero, then the TPer SHALL support a unique XTS-AES Tweak Key per Media Encryption Key that is provided by the host alongside the XTS-AES Key 1.

3.1.1.3.11 Incorrect Key Detection Supported

The Incorrect Key Detection Supported bit determines whether the Storage Device supports detection of when the key used for reads does not match the key that was used to write the data being read.

A Key Per I/O SSC compliant Storage Device MAY support Incorrect Key Detection.

If the Incorrect Key Detection Supported bit is set to one, then the Storage Device SHALL support detection of incorrect key usage for Interface Read commands.

If the Incorrect Key Detection Supported bit is cleared to zero, then the Storage Device SHALL NOT support detection of incorrect key usage for Interface Read commands.

3.1.1.3.12 Replay Protection Supported

The Replay Protection Supported field determines whether a TPer supports replay protection as part of Inject MEK or KEK requests using KMIP's operations (see sections 5.4.4.2.2.2.3, 5.4.4.2.3, and 5.4.4.2.4).

A Key Per I/O SSC compliant Storage Device MAY support Replay Protection.

If the Replay Protection Supported field is set to one, then the TPer SHALL support Replay Protection.

If the Replay Protection Supported field is cleared to zero, then the TPer SHALL NOT support Replay Protection.

If the Replay Protection Supported field is set to one, then the TPer SHALL support the Get Nonce command (see section 3.2.6).

3.1.1.3.13 Replay Protection Enabled

The Replay Protection Enabled field indicates whether Replay Protection has been enabled within a TPer.

If the Replay Protection Enabled field is set to one, then the TPer SHALL require that all encrypted symmetric key objects provisioned into a Namespace managed by Key Per I/O SSC are sent encrypted together with a Nonce value supplied by the TPer as specified by sections 3.2.6 and 4.3.5.1.4.

If the Replay Protected Enabled field is cleared to zero, then the TPer SHALL ignore the Nonce value if it is included in the wrapped key value and process the rest of the key injection operation normally (see section 4.3.5.1.4 for additional details).

3.1.1.3.14 Maximum Supported Key Unique Identifier Length

Start of informative comment

The Maximum Supported Key Unique Identifier Length is a value established by a Storage Device's manufacturer in order to limit the storage space needed on a Storage Device for managing the lookup of persistently stored keys such as KEKs.

In addition, features such as Incorrect Key Detection may use the MEKs' KeyUIDs to track key usage. For example, since a KeyUID is generally uniquely associated with a key within the identifier space managed by the key management system (see section 4.57 of [9] for details), manufacturers that implement the Incorrect Key Detection feature may leverage that uniqueness property and associate the MEK's KeyUID with the data written using that MEK on persistent media in order to detect incorrect MEK usage when that data is read (see section 5.5). In such cases, if the length of the KeyUID is more than the Maximum Supported Key Unique Identifier Length, truncated versions of the KeyUID could potentially lead to KeyUID collisions that would make the Incorrect Key Detection feature unreliable.

End of informative comment

The Maximum Supported Key Unique Identifier Length field is the maximum size in bytes supported by a TPer for a unique identifier value that is used to uniquely identify a key when used with key injection requests (see section 5.4.4.2).

If the Maximum Supported Key Unique Identifier Length is set to a non-zero value and the TPer receives a key injection operation that specifies a key object with a Unique Identifier whose length is larger than the value reported by this field, then the TPer SHALL fail this operation with Invalid Message Result Reason.

The TPer SHALL process the key injection operation normally if the specified key object's Unique Identifier length is less than or equal to the value reported by this field.

If the Maximum Supported Key Unique Identifier Length is set to zero, then the TPer SHALL process the key injection operation normally, with the length of the specified Unique Identifier for the key object being injected, constrained by the command payload's transfer size (see section 4.1.1.1.1).

3.1.1.3.15 KMIP Formatted Key Injection Supported

The KMIP Formatted Key Injection Supported bit indicates whether the Storage Device supports the KMIP format for injection of Key Encryption Keys and Media Encryption Keys. See sections 3.3.4 and 5.4 for more details on the KMIP format usage with TCG protocol and KMIP protocol elements usage for Key Per I/O SP respectively.

If the KMIP Formatted Key Injection Supported bit is set to one, then the TPer SHALL support parsing KMIP formatted Key Encryption Keys and Media Encryption Keys.

If the KMIP Formatted Key Injection Supported bit is cleared to zero, then the TPer SHALL NOT support parsing KMIP formatted Key Encryption Keys and Media Encryption Keys.

3.1.1.3.16 NIST AES-KW Supported

The NIST AES-KW Supported bit indicates whether the Storage Device supports the AES Key Wrap algorithm (defined in [11]) to be used as a key wrapping mechanism when injecting Key Encryption Keys and Media Encryption Keys.

If the NIST AES-KW Supported bit is set to one, then the TPer SHALL support unwrapping keys wrapped using the AES Key Wrap algorithm defined in [11].

If the NIST AES-KW Supported bit is cleared to zero, then the TPer SHALL NOT support unwrapping keys wrapped using the AES Key Wrap algorithm.

3.1.1.3.17 NIST AES-GCM Supported

The NIST AES-GCM Supported bit indicates whether the Storage Device supports the AES-GCM algorithm (defined in [12]) to be used as a key wrapping mechanism when injecting Key Encryption Keys and Media Encryption Keys.

If the NIST AES-GCM Supported bit is set to one, then the TPer SHALL support unwrapping keys wrapped using the AES-GCM algorithm defined in [12].

If the NIST AES-GCM Supported bit is cleared to zero, then the TPer SHALL NOT support unwrapping keys wrapped using the AES-GCM algorithm.

3.1.1.3.18 NIST RSA-OAEP Supported

The NIST RSA-OAEP Supported bit indicates whether the Storage Device supports the RSA-OAEP algorithm (defined in [13]) to be used as a key wrapping mechanism when injecting Key Encryption Keys as part of PKI-based KEK Transport (see section 3.1.1.3.24).

If the NIST RSA-OAEP Supported bit is set to one, then the TPer SHALL support unwrapping keys wrapped using the RSA-OAEP algorithm defined in [13].

If the NIST RSA-OAEP Supported bit is cleared to zero, then the TPer SHALL NOT support unwrapping keys wrapped using the RSA-OAEP algorithm.

3.1.1.3.19 AES-256 Wrapping Key Supported

The AES-256 Wrapping Key Supported bit indicates whether the Storage Device supports 256-bit key length for supported AES-based key wrapping algorithms.

If the AES-256 Wrapping Key Supported bit is set to one, then the TPer SHALL be capable of unwrapping keys wrapped with a 256-bit symmetric key.

If the AES-256 Wrapping Key Supported bit is cleared to zero, then the TPer SHALL NOT be capable of unwrapping keys wrapped with a 256-bit symmetric key.

3.1.1.3.20 RSA2K Wrapping Key Supported

The RSA2K Wrapping Key Supported bit indicates whether the Storage Device supports RSA 2048-bit key length in its RSA-OAEP key wrapping algorithm.

If the RSA2K Wrapping Key Supported bit is set to one, then the TPer SHALL be capable of unwrapping keys wrapped with a RSA-OAEP 2048-bit key.

If the RSA2K Wrapping Key Supported bit is cleared to zero, then the TPer SHALL NOT be capable of unwrapping keys wrapped with a RSA-OAEP 2048-bit key.

3.1.1.3.21 RSA3K Wrapping Key Supported

The RSA3K Wrapping Key Supported bit indicates whether the Storage Device supports RSA 3072-bit key length in its RSA-OAEP key wrapping algorithm.

If the RSA3K Wrapping Key Supported bit is set to one, then the TPer SHALL be capable of unwrapping keys wrapped with a RSA-OAEP 3072-bit key.

If the RSA3K Wrapping Key Supported bit is cleared to zero, then the TPer SHALL NOT be capable of unwrapping keys wrapped with a RSA-OAEP 3072-bit key.

3.1.1.3.22 RSA4K Wrapping Key Supported

The RSA4K Wrapping Key Supported bit indicates whether the Storage Device supports RSA 4096-bit key length in its RSA-OAEP key wrapping algorithm.

If the RSA4K Wrapping Key Supported bit is set to one, then the TPer SHALL be capable of unwrapping keys wrapped with a RSA-OAEP 4096-bit key.

If the RSA4K Wrapping Key Supported bit is cleared to zero, then the TPer SHALL NOT be capable of unwrapping keys wrapped with a RSA-OAEP 4096-bit key.

3.1.1.3.23 Plaintext Key Encryption Keys Provisioning Supported

The Plaintext Key Encryption Keys Provisioning Supported bit indicates whether the Storage Device supports injection of plaintext Key Encryption Keys or not.

If the Plaintext Key Encryption Keys Provisioning Supported bit is set to one, then the TPer SHALL support an option that allows the host to choose to provision any KEK in plaintext form (see section 4.3.5.1.5), and the TPer SHALL support an option that allows the host to selectively enable or disable Plaintext KEK provisioning on a per-KEK basis (see section 4.3.5.3.10)

If the Plaintext Key Encryption Keys Provisioning Supported bit is cleared to zero, then the TPer SHALL NOT support plaintext KEKs injection.

3.1.1.3.24 PKI-based Key Encryption Keys Transport Supported

The PKI-based KEK Transport Supported bit indicates whether the Storage Device supports injection of KEKs protected using RSA-OAEP or not.

If the PKI-based KEK Transport Supported bit is set to one, then the TPer SHALL support an option that allows the host to choose to provision any KEK protected using RSA-OAEP (see section 4.3.5.1.6), and the TPer SHALL support an option that allows the host to selectively enable or disable PKI-based KEK provisioning on a per-KEK basis (see section 4.3.5.3.10)

This bit SHALL be set to one for Storage Devices that support the RSA-OAEP wrapping algorithm.

If the PKI-based KEK Transport Supported bit is cleared to zero, then the TPer SHALL NOT support injection of RSA-OAEP protected KEKs.

3.1.1.3.25 Number of Key Encryption Keys Supported

The Number of Key Encryption Keys Supported field SHALL be the number of Key Encryption Keys that can be stored by a TPer at any given time.

3.1.1.3.26 Total Number of Key Tags Supported

The Total Number of Key Tags Supported field SHALL be the number of MEKs that can be injected and addressed at any given time in a TPer.

3.1.1.3.27 Maximum Number of Key Tags Supported Per Namespace

Start of informative comment

The Maximum Number of Key Tags Supported Per Namespace field is the maximum number of key tags per Namespace that the Storage Device supports. It is a value determined by a Storage Device manufacturer and it is informed by the context of the manufacturer's Storage Devices' key lookup engine design and not necessarily the size

of its key cache. Because key lookup is most likely performed during a critical I/O path, a theoretical maximum serves to inform how manufacturers optimize their key lookup functionality.

In use cases where the ability to dynamically configure the number of key tags available for a Namespace is critical, Key Per I/O SSC defines a host-settable, dynamically configurable limit of the number of key tags that are allocated per Namespace at any given time (see sections 3.1.2.5.5 and 4.3.5.2.6 for more details). This dynamically configurable value should not exceed the value indicated by the Maximum Number of Key Tags Supported Per Namespace field.

End of informative comment

The Maximum Number of Key Tags Supported Per Namespace field SHALL be the maximum number of Key Tags that can be allocated to any Namespace via the NumberOfKeyTags column in the `KeyTagAllocation` Table (see section 4.3.5.2.6) that the Storage Device supports.

3.1.1.3.28 Get Nonce Command Nonce Length

The Get Nonce Command Nonce Length field SHALL be the fixed length in bytes of the Nonce value that the TPer supports for the Get Nonce Command (see section 3.2.6).

3.1.1.3.29 Key Per I/O SSC Feature Descriptor Requirements

A Key Per I/O SSC compliant Storage Device SHALL return the following field values in its Key Per I/O SSC Feature Descriptor:

- Feature Code = 0x0305
- Feature Descriptor Version Number = 0x1 or any version that supports the defined features in this SSC
- SSC Minor Version Number = As specified in Table 5
- Length = 0x2C
- Protocol 0x01 Base ComID = VU
- Protocol 0x01 Number of ComIDs = 0x0001 or larger
- Protocol 0x03 Base ComID = VU
- Protocol 0x03 Number of ComIDs = 0x0001 or larger
- Initial C_PIN_SID PIN Indicator = VU
  - 0x00 = The initial C_PIN_SID PIN value is equal to the C_PIN_MSID PIN value
  - 0xFF = The initial C_PIN_SID PIN value is VU, and SHALL NOT be equal to the C_PIN_MSID PIN value
  - 0x01 – 0xFE = Reserved
- Behavior of C_PIN_SID PIN upon TPer Revert = VU
  - 0x00 = The C_PIN_SID PIN value becomes the value of the C_PIN_MSID `PIN` column after successful invocation of `Revert` on the Admin SP's object in the `SP` table
  - 0xFF = The C_PIN_SID PIN value changes to a VU value after successful invocation of `Revert` on the Admin SP's object in the `SP` table, and SHALL NOT be equal to the C_PIN_MSID PIN value
  - 0x01 – 0xFE = Reserved

Table 5: SSC Minor Versions

Key Per I/O SSC Minor Version	Standard Referenced
0x00	TCG Key Per I/O SSC Specification v1.00
All others	Reserved

- Number of Key Per I/O SP Admin Authorities Supported = 0x0001 or larger
- Replay Protection Enabled = See sections 3.1.1.3.13 and 4.3.5.1.4

- Replay Protection Supported = VU
- Incorrect Key Detection Supported = VU
- Shared XTS-AES Tweak Key Required = VU
- Key Per I/O Enabled = See section 3.1.1.3.8
- Key Per I/O Scope = VU
- Maximum Supported Key Unique Identifier Length = VU
- KMIP Formatted Key Injection Supported = 0x0001
- NIST RSA-OAEP Supported = VU
- NIST AES-GCM Supported = See section 2.5.
- NIST AES-KW Supported = See section 2.5.
- AES-256 Wrapping Key Supported = 1b
- RSA4K Wrapping Key Supported = VU
- RSA3K Wrapping Key Supported = VU
- RSA2K Wrapping Key Supported = VU
- Plaintext KEK Provisioning Supported = VU
- PKI-based KEK Transport Supported = VU
- Number of Key Encryption Keys Supported = 0x0001 or larger
- Total Number of Key Tags Supported = 0x0001 or larger
- Maximum Number of Key Tags Supported Per Namespace = 0x0001 or larger
- Get Nonce Command Nonce Length = VU
- Reserved fields SHALL be set to 0s.

3.1.1.4 Supported Data Removal Mechanism Feature (Feature Code = 0x0404)

Table 6: Level 0 Discovery – Supported Data Removal Mechanism Feature Descriptor

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)							
1	Feature Code (0x0404)							(LSB)
2	Version				Reserved			
3	Length							
4	Reserved							
5	Reserved						Data Removal Operation Interrupted	Data Removal Operation Processing
6	Supported Data Removal Mechanism							
7	Reserved		Data Removal Time Format for Bit 5	Data Removal Time Format for Bit 4	Data Removal Time Format for Bit 3	Data Removal Time Format for Bit 2	Data Removal Time Format for Bit 1	Data Removal Time Format for Bit 0
8-9	Data Removal Time for Supported Data Removal Mechanism Bit 0							
10-11	Data Removal Time for Supported Data Removal Mechanism Bit 1							
…..	…….							
18-19	Data Removal Time for Supported Data Removal Mechanism Bit 5							
20-35	Reserved for future Supported Data Removal Mechanism parameters							

A Key Per I/O SSC compliant Storage Device SHALL return the following field values:

- Feature Code = 0x0404
- Version = 0x1 or any version that supports the defined features in this SSC
- Length = 0x20
- Data Removal Operation Processing = See section 3.1.1.4.1
- Data Removal Operation Interrupted = See section 3.1.1.4.2
- Supported Data Removal Mechanism = VU, See section 3.1.1.4.3
- Data Removal Time for Supported Data Removal Mechanism Bit = See section 3.1.1.4.4
- Reserved = 0x0000

3.1.1.4.1 Data Removal Operation Processing Definition

The Data Removal Operation Processing bit SHALL be set to one if a TPer is performing a `Revert` operation; otherwise, the Data Removal Operation Processing bit SHALL be set to zero. If the operation is in progress, the security transport commands such as the Security Send, and the Security Receive SHALL be processed by the Storage Device. The Data Removal Operation Processing bit SHALL be cleared to zero upon completion of a data removal operation.

The Data Removal Operation Processing bit SHALL be set to one if the data removal operation is restarted after a Power Cycle (see Table 25).

3.1.1.4.2 Data Removal Operation Interrupted

The Data Removal Operation Interrupted bit SHALL be set to one if a previously issued `Revert` was interrupted for any reason (including power loss, interface reset, etc.). The Data Removal Operation Interrupted bit SHALL be cleared to zero after completion of a data removal operation.

3.1.1.4.3 Supported Data Removal Mechanism Definition

Each bit of the Supported Data Removal Mechanism (see Table 7) SHALL be set to one if a TPer supports the corresponding Data Removal Mechanism; otherwise, each bit SHALL be set to zero. The TPer SHALL support Overwrite Data Erase or Block Erase. The TPer MAY support Crypto Erase mechanism and may support other mechanisms. The TPer MAY support multiple Data Removal Mechanisms described in Table 7. After a `Revert` method has completed without an error, the condition of user data SHALL be indicated as specified in Table 7.

Table 7: Supported Data Removal Mechanism

Bit	Name	Condition of user data after Data Removal
0	Overwrite Data Erase[1]	The Overwrite Data Erase mechanism SHALL cause the TPer to alter information by writing a vendor specific data pattern to the medium.
1	Block Erase[1]	The Block Erase mechanism SHALL cause the TPer to alter information by setting the physical blocks to a vendor specific value.
2	Cryptographic Erase[2]	A TPer MAY support this data erasure mechanism via the `Revert` method. The Crypto Erase mechanism SHALL eradicate the cryptographic keys located on a Storage Device that are used to encrypt the user data in a way that the user data remains irretrievable even if the same cryptographic keys are reused (i.e., re-injected) by the host after `Revert` method has completed successfully.
3	Unmap	The Unmap mechanism SHALL cause all mapping between LBAs and internal media to be removed.
4	Reset Write Pointers	The Reset Write Pointers mechanism SHALL cause all zones (i.e., write pointer zones) to have the write pointer set to the first LBA in that zone.
5	Vendor Specific Erase[1]	The Vendor Specific Erase mechanisms SHALL cause all user data to be removed by a vendor specific method. [3]
6-7	Reserved	

Notes:

[1] If the Cryptographic Erase data removal mechanism is supported, then it SHALL also be performed when this data removal mechanism is used.

[2] The Cryptographic Erase operation MAY be used by the `Revert` method. Any subsequent operation(s) such as Deallocate, or Unmap, or Trim, that is part of the implementation of the data removal operation SHALL be accounted for in the time reported for this operation (see section 3.1.1.4.4). The time value reported SHALL correspond to an estimated completion time of the Cryptographic Erase.

[3] If an Storage Device supports more than one vendor proprietary method of data removal, then the associated estimated time value SHALL represent the completion time for the longest of the vendor specific mechanisms.

3.1.1.4.4 Data Removal Time Format and Data Removal Time Definition

Each Data Removal Time field provides the worst-case estimate of the time required to perform the erasure corresponding to each Data Removal Mechanism defined in the Supported Data Removal Mechanism field. The Data Removal Time Format bit identifies the format used to express the time as follows:

a) if the Data Removal Time Format bit is set to zero, then the estimated time is defined in Table 8; and

b) if the Data Removal Time Format bit is set to one, then the estimated time is defined in Table 9.

The Data Removal Time Format bit and Data Removal Time Format field are defined in Table 8 and Table 9.

Table 8: Data Removal Time (Data Removal Time Format bit= 0)

Value	Time
0	Not reported
1..65534	(Value x 2) seconds
65535	>= 131068 seconds

Table 9: Data Removal Time (Data Removal Time Format bit= 1)

Value	Time
0	Not reported
1..65534	(Value x 2) minutes
65535	>= 131068 minutes

Start of informative comment

Each Data Removal Time field is an estimate of the total time required to perform the erasure for each corresponding Data Removal Mechanism. This field is not a dynamic estimate of the remaining time for completion.

End of informative comment

3.1.2 Namespace Level 0 Discovery

3.1.2.1 Overview

The Namespace Level 0 Discovery command provides a host with basic information about TPer capabilities both current and potential, for a specific Namespace.

3.1.2.2 IF-SEND Command

There is no IF-SEND command defined for Namespace Level 0 Discovery. So, if IF-SEND is invoked with a Protocol ID of 0x01 and a ComID of 0x0002, then the TPer SHALL:

1. transfer all of the data from the host;
2. discard the data; and
3. return ‘good’ status to the host.

3.1.2.3 IF-RECV Command

Namespace Level 0 Discovery is invoked by sending an IF-RECV command with:

Protocol ID = 0x01
ComID = 0x0002
Transfer Length = maximum length of the Namespace Level 0 Discovery response data that the host elects

to receive.
NamespaceID = identifier for a Namespace/LUN.

The Namespace Level 0 Discovery IF-RECV command may be processed at any time, without regard to sessions or prior authentication.

If the Transfer Length parameter is less than the size of the Namespace Level 0 Discovery response data that is available, then the TPer SHALL return the requested amount of data, even if it is truncated.

If the Transfer Length parameter is greater than the size of the Namespace Level 0 Discovery response data, then the Storage Device SHALL pad according to the rules specified in the transport.

If the NamespaceID parameter [5] specifies:

a) an allocated Namespace/LUN identifier (i.e., a value that corresponds to an existing Namespace/LUN), then the TPer SHALL return Namespace Level 0 Discovery Response Data containing feature descriptors corresponding to that Namespace/LUN;

b) a value of 0xFFFF_FFFF, then the TPer SHALL return the Namespace Level 0 Discovery header and zero feature descriptors; and

c) any other value, then the TPer SHALL fail the command with a status of Other Invalid Command Parameter.

The Namespace Level 0 Discovery response data (see Table 10) consists of a header field and zero or more variable length feature descriptors. A TPer SHALL NOT include feature descriptors for Namespace/LUN features that it does not implement. The data is not packetized.

Table 10: Namespace Level 0 Discovery Response Data Format

Bit / Byte	7	6	5	4	3	2	1	0
0 – 47	Namespace Level 0 Discovery Header (see Table 11)							
48 – n	Feature Descriptor(s) (see section 3.1.2.4)							

Table 11: Namespace Level 0 Discovery Header Format

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)							
		Length of Parameter Data						
3								(LSB)
4	(MSB)							
		Data Structure Revision						
7								(LSB)
8								
		Reserved						
47								

3.1.2.3.1 Length of Parameter Data

The Length of the Parameter Data field is the total number of bytes that are valid in the Namespace Level 0 Discovery header and all of the feature descriptors returned, excluding the Length of the Parameter Data field itself. If no feature descriptors are returned, then this field SHALL be set to 0x0000_002C.

3.1.2.3.2 Data Structure Revision

The Data Structure Revision field describes the format of the Namespace Level 0 Discovery header returned. The value SHALL be 0x0000_0001.

3.1.2.4 Namespace Level 0 Discovery Feature Descriptors

The Namespace Feature Descriptors SHALL be returned in the Namespace Level 0 Discovery response data in order of increasing Namespace/LUN feature code values. Namespace features that are not implemented SHALL NOT be returned.

Table 12 contains the feature code for Namespace Level 0 Discovery.

Table 12: Namespace Level 0 Discovery Feature Codes

Feature Code	Feature Name	Description
0x040A	Namespace Key Per I/O Capabilities Feature	See section 3.1.2.5

3.1.2.5 Namespace Key Per I/O Capabilities Feature (Feature Code = 0x040A)

The Namespace Key Per I/O Capabilities Feature (see Table 13) indicates the various Key Per I/O capabilities supported within the Namespace/LUN indicated by the NamespaceID parameter of the IF-RECV command. The Namespace Key Per I/O Capabilities Feature Descriptor feature SHALL be returned in the Namespace Level 0 Discovery response.

Table 13: Level 0 Discovery – Namespace Key Per I/O Capabilities Feature Descriptor

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)	Namespace Key Per I/O Capabilities Feature Code (0x040A)						
1								(LSB)
2	Version				Reserved			
3	Length							
4								Managed By Key Per I/O
5-6	Number of Allocated Key Tags							
7	(MSB)	Reserved						
…								
11								(LSB)
12	(MSB)	Reserved						
…								
15								(LSB)
16	(MSB)	Reserved						
…								
23								(LSB)
24	(MSB)	Reserved						
…								
31								(LSB)

3.1.2.5.1 Namespace Key Per I/O Capabilities Feature Code

The Namespace Key Per I/O Capabilities Feature Code field SHALL be set to 0x040A.

3.1.2.5.2 Version

The Version field is 0x1 or any version that supports the features described in this specification.

3.1.2.5.3 Length

The Length field is the number of bytes in the descriptor following byte 3. The value SHALL be set to `0x1C`.

3.1.2.5.4 Managed By Key Per I/O

The Managed By Key Per I/O field determines whether the Namespace specified via the NamespaceID field in the Namespace Level 0 Discovery command is currently managed by Key Per I/O.

Start of informative comment

The value of the Managed By Key Per I/O field is equal to the value of the Managed column value for the KeyTagAllocation table row that is associated with the NamespaceID that was specified in the Namespace Level 0 Discovery command:

If Managed = True, then Managed By Key Per I/O = 1

If Managed = False, then Managed By Key Per I/O = 0

End of informative comment

If the `KeyTagAllocation` Table row with a NamespaceID column value equal to the NamespaceID field in the Namespace Level 0 Discovery command has a Managed column value equal to True, then the Managed By Key Per I/O field SHALL be set to one.

If the `KeyTagAllocation` Table row with a NamespaceID column value equal to the NamespaceID field in the Namespace Level 0 Discovery command has a Managed column value equal to False, then the Managed By Key Per I/O field SHALL be cleared to zero.

3.1.2.5.5 Number of Allocated Key Tags

The Number of Allocated Key Tags field SHALL be the number of key tags currently allocated by the host for the Namespace. This value SHALL be equal to the value indicated by the NumberOfKeyTags column of the `KeyTagAllocation` Table row that is associated with the NamespaceID that was specified in the Namespace Level 0 Discovery command

3.1.2.5.6 Namespace Key Per I/O Capabilities Feature Descriptor Requirements

A Key Per I/O SSC compliant Storage Device SHALL return the following field values:

- Namespace Key Per I/O Capabilities Feature Code = 0x040A
- Version = 0x1 or any version that supports the defined features in this SSC
- Length = 0x1C
- Managed By Key Per I/O = See section 4.3.5.2.5 for details.
- Number of Allocated Key Tags = See section 4.3.5.2.6 for details.
- Reserved fields SHALL be set to 0s.

3.2 Security Protocol 2 Support

3.2.1 ComID Management

ComID management support is reported in Level 0 Discovery. Statically allocated ComIDs are also discoverable via the Level 0 Discovery response.

3.2.2 Stack Protocol Reset (M)

A Key Per I/O SSC compliant Storage Device SHALL support the Stack Protocol Reset command. Refer to [3] for details.

3.2.3 TPER_RESET command (M)

If the TPER_RESET command is enabled, it SHALL cause the following before a TPer accepts the next IF-SEND or IF-RECV command:

a) all dynamically allocated ComIDs SHALL return to the Inactive state;
b) all open sessions SHALL be aborted on all ComIDs;
c) all uncommitted transactions SHALL be aborted on all ComIDs;
d) the synchronous protocol stack for all ComIDs SHALL be reset to its initial state;
e) all TCG command and response buffers SHALL be invalidated for all ComIDs;
f) all related method processing occurring on all ComIDs SHALL be aborted;
g) the TPer's knowledge of the host's communications capabilities, on all ComIDs, SHALL be reset to the initial minimum assumptions defined in [3] or the TPer's SSC definition, and
h) the values of the AccessLocked column SHALL be set to True for all Key Per I/O SP's Key Encryption Key objects that contain the Programmatic enumeration value in the LockOnReset column.

The TPER_RESET command is delivered by the transport IF-SEND command. If the TPER_RESET command is enabled, the TPer SHALL accept and acknowledge it at the interface level. If the TPER_RESET command is disabled, the TPer SHALL abort it at the interface level with the "Other Invalid Command Parameter" status (see [5]). There is no IF-RECV response to the TPER_RESET command.

The TPER_RESET command is defined in Table 14.

The Transfer Length SHALL be non-zero. All data transferred SHALL be ignored.

Table 14: TPER_RESET Command

FIELD	VALUE
Command	IF-SEND
Protocol ID	0x02
Transfer Length	Non-zero
ComID	0x0004

3.2.4 Clear Single MEK Command (M)

3.2.4.1 Command Overview

Start of informative comment

The Clear Single MEK command and data structure payload provides the caller with a mechanism to clear a single MEK from a TPer's key cache.

This operation does not affect any user data associated with the MEK.

If the same MEK is injected again, it can once again be used to read data that was previously written and encrypted using the associated MEK.

Prior to issuing the Clear Single MEK command, the host needs to ensure that it has no pending I/O associated with the MEK it wants to clear. Otherwise, the host risks errors on pending I/O associated with the MEK that is being removed (see section 5.5.1[5]). If there is pending I/O, it is recommended that the host waits for the I/O associated with the Namespace to complete before issuing the Clear Single MEK command to that Namespace.

The Clear Single MEK command does not affect Key Tag Allocation settings configured in the `KeyTagAllocation` table.

End of informative comment

The Clear Single MEK Slot command SHALL clear the key associated with the Key Tag Value from a TPer's key cache.

If the Key Per I/O SP is not in a Manufactured Life Cycle state, then the Clear Single MEK command SHALL fail with status Operation Denied.

If the value specified in the Namespace ID field:

1) Does not match an existing Namespace;
2) Is 0x00000000; or
3) Is 0xFFFFFFFF

then, the IF command that specifies it SHALL fail with a status Other Invalid Command Parameter (see [5]).

If the Namespace that is specified by the Namespace ID field is not managed by the Key Per I/O SP, then the Clear Single MEK command SHALL fail with the Not Key Per I/O Managed error status (see Table 16 for details on protocol error status).

If the specified Key Tag is greater than or equal to the NumberOfKeyTags column value for the Namespace identified by the specified NSID value, then the command SHALL fail with Invalid Key Tag error status (see Table 16 for details on protocol error status).

If the column value of the ClearSingleMEKAllowed column in the `KPIOPolicies` Table (see 4.3.5.1) is set to False, then the command SHALL fail with CmdLocked error status (see Table 16 for details on protocol error status).

The Clear Single MEK command is delivered in the payload of the HANDLE_COMID_REQUEST command (see [3] for details on HANDLE_COMID_REQUEST).

The response SHALL be returned via the GET_COMID_RESPONSE (IF-RECV) command (see [3] details on GET_COMID_RESPONSE).

The Clear Single MEK command is defined in Table 15.

Table 15: Clear Single MEK Command Request

Bytes	Field	Value
0 to 3	Extended ComID	Allocated ComID
4 to 7	Request Code	00 00 00 03
8 to 9	Key Tag	Key Tag value
10 to TRNSFLEN-1	Reserved	00

The Clear Single MEK command response payload is defined in Table 16.

Table 16: Clear Single MEK Command Response Payload

Byte	Field	Value
0 to 3	Extended ComID	Allocated ComID
4 to 7	Request Code	00 00 00 03
8 to 9	Reserved	00 00
10 to 11	Available Data Length in Bytes	00 04
12 to 15	Status	00 00 00 00 = Success, 00 00 00 01 = Failure, 00 00 00 02 = CmdLocked, 00 00 00 03 = Invalid Key Tag, 00 00 00 04 = Not Key Per I/O Managed
16 to TRNSFLEN-1	Reserved	00

The response payload status values in Table 16 are defined as follows:

- Success(`0x00000000`) SHALL indicate that the MEK associated with the specified Key Tag has been cleared.
- Failure (`0x00000001`) SHALL indicate that the MEK associated with the specified Key Tag has not been changed due to a Storage Device's internal errors.
- CmdLocked(`0x00000002`) SHALL indicate that the command was rejected because it is disabled by the `KPIOPolicies` Table (see section 4.3.5.1.3 for details on the policy definition).
- Invalid Key Tag (`0x00000003`) SHALL indicate that the specified Key Tag value is not associated with an injected MEK.
- Not Key Per I/O Managed (`0x00000004`) SHALL indicate that the Namespace specified by the NSID field value is not being managed by Key Per I/O SP (see section 4.3.5.2.5 for details on Managed Namespaces).

The Storage Device SHALL return "No Response Available" if No HANDLE_COMID_REQUEST preceded the GET_COMID_RESPONSE command.

The "No Response Available" payload is defined in [3]

The Storage Device SHALL return “Pending” if the host retrieves the command result via the GET_COMID_RESPONSE command while Clear Single MEK command is in progress for that specific ComID.

The “Pending” payload is defined in Table 17 .

Table 17: Clear Single MEK Pending

Byte	Field	Value
0 to 3	Extended ComID	Allocated ComID
4 to 7	Request Code	00 00 00 03
8 to 9	Reserved	00 00
10 to 11	Available Data Length in Bytes	00 00
12 to TRNSFLEN-1	Reserved	00

3.2.5 Clear All MEKs command (M)

3.2.5.1 Command Overview

Start of informative comment

The Clear All MEKs command and data structure payload provide the caller with a mechanism to clear all MEKs in the Storage Device key cache assigned to the Namespace identified by the specified NSID value.

This operation does not affect any user data associated with the MEKs that were cleared.

If the same MEK is injected again, it can once again be used to read data that was previously written and encrypted using the associated MEK.

Prior to issuing the Clear All MEKs command, the host needs to ensure that it has no pending I/O associated with the MEKs it wants to clear. Otherwise, the host risks errors on pending I/O associated with the MEKs that are being removed (see section 5.5.1[5]). If there is pending I/O, it is recommended that the host waits for the I/O associated with the Namespace to complete before issuing the Clear All MEKs command to that Namespace.

The Clear All MEKs command does not affect Key Tag Allocation settings configured in the `KeyTagAllocation` Table.

End of informative comment

The Clear All MEKs command SHALL clear all keys associated with the Namespace identified by the NSID value from a TPer’s key cache.

If the Key Per I/O SP is not in a Manufactured Life Cycle state, then the Clear All MEKs command SHALL fail with status Operation Denied.

If the value specified in the Namespace ID field:

1) Does not match an existing Namespace; or
2) Is 0x00000000.

Then, the IF command that specifies it SHALL fail with a status Other Invalid Command Parameter (see [5]).

If the value specified in the NSID field is 0xFFFFFFFF, the TPer SHALL clear all MEKs associated with all Namespaces managed by Key Per I/O that are configured with a non-zero number of key tags from the TPer’s key cache.

If the Namespace that is specified by the NSID field is not managed by Key Per I/O SP, then the Clear All MEKs command SHALL fail with the Not Key Per I/O Managed error status (see Table 19 for details on protocol error status).

If the column value of the ClearAllMEKsAllowed column in the `KPIOPolicies` Table is set to False, then the command SHALL fail with CmdLocked error status (see Table 19 for details on protocol error status).

The Clear All MEKs command is delivered in the payload of the HANDLE_COMID_REQUEST command (see [3] for details on HANDLE_COMID_REQUEST).

The response SHALL be returned via the GET_COMID_RESPONSE (IF-RECV) command (see [3] for details on GET_COMID_RESPONSE).

The Clear All MEKs command payload is defined in Table 18.

Table 18: Clear All MEKs Command Request

Bytes	Field	Value
0 to 3	Extended ComID	Allocated ComID
4 to 7	Request Code	00 00 00 04
8 to TRNSFLEN-1	Reserved	00

The Clear All MEKs command response payload is defined in Table 19.

Table 19: Clear All MEKs Command Response Payload

Byte	Field	Value
0 to 3	Extended ComID	Allocated ComID
4 to 7	Request Code	00 00 00 04
8 to 9	Reserved	00 00
10 to 11	Available Data Length in Bytes	00 04
12 to 15	Status	00 00 00 00 = Success, 00 00 00 01 = Failure, 00 00 00 02 = CmdLocked, 00 00 00 03 = Reserved, 00 00 00 04 = Not Key Per I/O Managed
16 to TRNSFLEN -1	Reserved	00

The response payload status values in Table 19 are defined as follows:

- Success(`0x00000000`) SHALL indicate that the MEKs associated with the specified Namespace ID have been cleared.

- Failure (0x00000001) SHALL indicate that the MEKs associated with the specified Namespace ID have not been changed due to a Storage Device's internal errors.
- CmdLocked(0x00000002) SHALL indicate that the command was rejected because it is disabled by the KPIOPolicies Table (see section 4.3.5.1.3 for details on the policy definition).
- Not Key Per I/O Managed (0x00000004) SHALL indicate that the Namespace specified by the NSID field value is not being managed by Key Per I/O SP (see section 4.3.5.2.5 for details on Managed Namespaces).

The Storage Device SHALL return "No Response Available" if No HANDLE_COMID_REQUEST preceded the GET_COMID_RESPONSE command.

The "No Response Available" payload is defined in [3].

The Storage Device SHALL return "Pending" if the host retrieves the command result via the GET_COMID_RESPONSE command while Clear All MEKs command is in progress for that specific ComID.

The "Pending" payload is defined in Table 20.

Table 20: Clear All MEKs Pending

Byte	Field	Value
0 to 3	Extended ComID	Allocated ComID
4 to 7	Request Code	00 00 00 04
8 to 9	Reserved	00 00
10 to 11	Available Data Length in Bytes	00 00
12 to TRNSFLEN-1	Reserved	00

3.2.6 Get Nonce Command (O)

Start of informative comment

The Get Nonce command is an IF-RECV command that is defined for Security Protocol 0x02 for the purpose of replay protection feature (see section 3.1.1.3.12). The command gives a TPer the ability to issue a challenge (i.e., Nonce) to the host that the host can then use to demonstrate back to the TPer the current proof of possession of the key material being injected, as opposed to a malicious entity replaying an old copy of the same key material.

Current proof of possession of the key material is reliable only if the key material and the Nonce value are transmitted to a TPer encrypted together via authenticated encryption (see [12] for example) that encompasses both the Nonce and the key material. This ensures that each component of the encrypted data cannot be independently replaced and replayed without detection during transmission.

See section 5.4.4.2.2.2.3 for details on how a host specifies the Nonce value retrieved from a Storage Device in the request to retrieve a wrapped key from the key management server.

End of informative comment

The Get Nonce command can be used by the host to request a Nonce to be used for the next request to inject MEKs or KEKs symmetric key objects for a specific NamespaceID.

The Get Nonce Command SHALL be supported if a Storage Device supports Replay Protection as specified in the Anti-Replay Protection Supported field in the Key Per I/O SSC Feature Descriptor (see section 3.1.1.3.12).

The Get Nonce command block and response payload are defined in Table 21 and Table 22 respectively.

Table 21: Get Nonce Command Block

Field	Value
Command	IF-RECV
Protocol ID	0x02
Transfer Length	Non-zero
ComID	0x0006
NSID	Namespace ID

Table 22: Get Nonce Response Payload

Byte	Description
0 – (NL-1)	Nonce

If Replay Protection is enabled within the TPer as indicated by the `KPIOPolicies` Table (see section 4.3.5.1) and Level 0 Discovery (see section 3.1.1.3.13), the TPer SHALL process Get Nonce command requests.

The TPer SHALL track Nonce usage for each key injection request. Once a Nonce has been used by the key injection request, the TPer SHALL stop tracking it and discard it. A Nonce may be reused by multiple key injection KMIP Import operations within the same key injection request but it cannot be reused across multiple key injection requests (see section 4.3.5.1.4 for additional details).

The TPer MAY discard skipped Nonces to make room for fresh Nonces in order to prevent blocking of its ability to return fresh Nonces once all have been requested. The TPer SHALL discard the Nonces it is tracking if it experiences a power cycle.

The Nonce Length (NL) is vendor implementation specific and is discoverable via Key Per I/O's Feature Descriptor (see 3.1.1.3.28). The Get Nonce command SHALL return a Nonce value that is composed of a sequence of random bytes. The definition of the quality of random numbers generated is outside the scope of this specification.

If Replay Protection is disabled within the TPer as indicated by the `KPIOPolicies` Table (see 4.3.5.1) and Level 0 Discovery (see 3.1.1.3.13), the TPer SHALL fail Get Nonce command requests with status Operation Denied (see [5]).

3.3 Security Protocol 3 Support

Start of informative comment

The following section describes the communication infrastructure for Security Protocol 0x3 interactions between the TPer and the host entities using the Key Management Interoperability Protocol (KMIP), a communication protocol leveraged by Key Per I/O SSC for formatting key material and transporting it from the host into a Storage Device.

This section defines how ComIDs are managed for this protocol and how they interact with other security protocols supported by Key Per I/O SSC. In addition, this section also specifies how the KMIP Request Response Message infrastructure is encoded onto the existing TCG storage communication protocol.

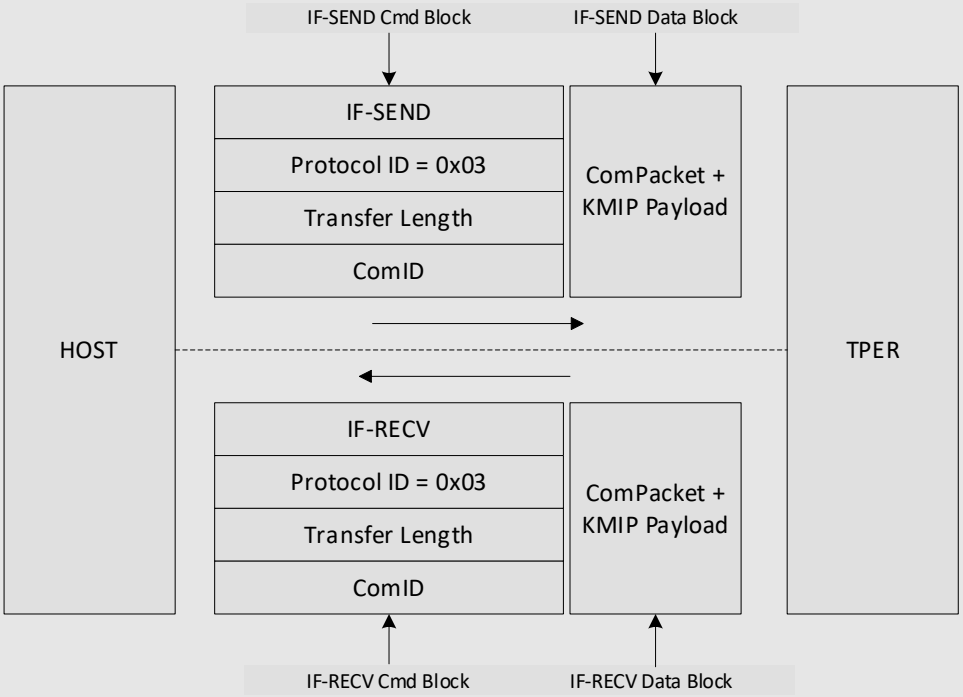
End of informative comment

3.3.1 TPer – Host Communication Over Protocol ID 0x03

Start of informative comment

As Figure 9 illustrates, consistent with the TCG communication interface protocol, the TPer – KMIP communication over Protocol 0x3 utilizes a ComID to identify the caller of the IF-SEND / IF-RECV command. The IF command's data block payload is used to convey KMIP protocol data encoded as the payload of the ComID's ComPacket (see section 3.3.4 for details).

Figure 9: TPer - Host Communication for Protocol ID 0x03



The host requests a ComID from the TPer if it does not already have one. The ComID can be either static or dynamic. With statically allocated ComIDs, the host can pick one and immediately begin using it as these ComIDs are always in an "Issued" state (see [3] for the TCG ComID states definitions). With dynamically allocated ComIDs, the host queries the TPer for available ComID using the GET_COMID command over Protocol ID 0x02 (see [3] section 3.3.4.3.1 for details on the usage of the GET_COMID command). The TPer chooses from its pool of "Inactive" ComIDs and then issues a ComID to the host application and transitions this ComID's state into the "Issued" state. At this point, the host is expected to use this ComID in its subsequent communication to the TPer. Support for multiple ComIDs allows for multiple host applications to simultaneously communicate with the TPer without interfering with one another.

As described in [3], a dynamically allocated ComID in the "Issued" state is assigned to its requestor until 1) the next reset event occurs (which resets all ComIDs to their initial "Inactive" state) or 2) if the requestor does not use the ComID within the MaxComIDTime from the ComID being issued (see [3] for the definition of MaxComIDTime). In the event that the TPer is unable to assign a new ComID, it returns all 0s to the host. The latter can try requesting a ComID again and again, until it succeeds.

With statically allocated ComIDs, since they are always in the “Issued” state, the host is generally responsible for ensuring that ComID allocation is properly synchronized among the host’s applications.

Finally, ComIDs in use over Protocol 0x03 do not need to be in an “Associated” state (see [3]) as sessions are not supported over this Security Protocol.

End of informative comment

3.3.2 ComID Management

ComID management support is reported in Level 0 Discovery. Statically allocated ComIDs are also discoverable via the Level 0 Discovery response.

3.3.3 ComID Binding to Security Protocol

Start of informative comment

To enable flexible support for dynamic ComID over the various security protocols supported by Key Per I/O SSC, this specification defines a runtime mapping of a dynamically allocated ComID that binds it to the security protocol onto which it is first used. Once the binding has been established, it lasts until it is cleared by a ComID protocol stack reset event such as TCG defined SIIS reset events or TCG-defined reset commands such as a Stack reset or TPer reset.

Statically allocated ComIDs are statically pre-allocated to the security protocols in which they will be used and thus do not support the concept of runtime binding. Their binding, as indicated by Level 0 Discovery, is static and persists across all reset events.

End of informative comment

If the TPer receives an IF-SEND or IF-RECV command where:

a) the specified Protocol ID is either 0x01 or 0x03; and
b) the specified ComID is not bound to either Protocol ID 0x01 or 0x03,

then the TPer SHALL bind that ComID to the specified Protocol ID.

If the TPer receives an IF-SEND or IF-RECV command where:

a) the specified ComID is bound to one Protocol ID; and
b) the specified Protocol ID does not match the Protocol ID to which the specified ComID is bound,

then the IF command SHALL fail with a status of Invalid Security Protocol ID Parameter.

3.3.3.1 TCG Resets and ComID Binding

Start of informative comment

The following rules apply to TCG-defined reset commands and TCG reset events defined in the SIIS specification.

End of informative comment

if the protocol stack for a dynamically allocated ComID is reset, then the binding of this ComID to a specific Security Protocol SHALL be cleared.

if the protocol stack for a statically allocated ComID is reset, then the binding of this ComID to a specific Security Protocol SHALL NOT be changed.

3.3.3.2 Security Protocol 0x02 and ComID Binding

The TPer SHALL accept communications on Security Protocol 0x02 for ComIDs that are bound to Security Protocol 0x01 or 0x03.

3.3.4 Security Protocol 3 Payload

Start of informative comment

Security Protocol 0x03 is used to support the ability to dynamically inject keys into a Storage Device using the KMIP protocol. This subsection defines how the KMIP Request – Response Message protocol is encoded onto the TCG communication protocol in a manner that preserves compatibility with the TCG synchronous communication protocol (as defined in [3] section 3.3.10.4) on Security Protocol 0x03.

End of informative comment

The TPer SHALL support the Key Management Interoperability Protocol (KMIP) on Security Protocol 0x03.

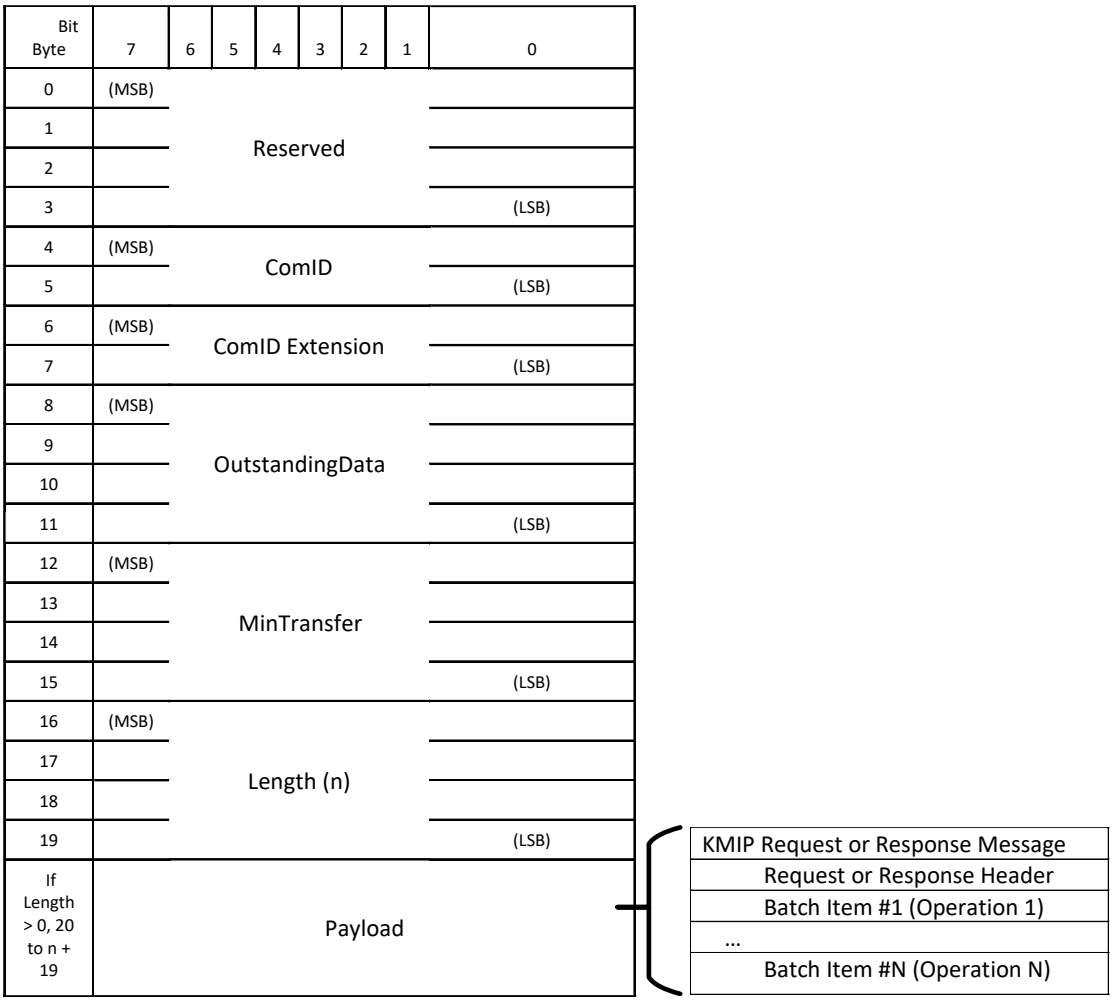
The TPer SHALL support a modified ComPacket as the data block for the IF-SEND and IF-RECV commands for Security Protocol 0x03.

The ComPacket payload for IF-SEND commands for Security Protocol 0x03 SHALL be a KMIP Request Message as shown in Figure 10.

The ComPacket payload for IF-RECV commands for Security Protocol 0x03 SHALL be a KMIP Response Message as shown in Figure 10.

If the Key Per I/O SP is not in a Manufactured Life Cycle state, then the IF command received by the TPer SHALL FAIL with status Invalid Security Protocol ID Parameter.

Figure 10: TCG ComPacket with KMIP Request or Response Message



For details on the minimum set of the KMIP protocol elements (KMIP operations, objects, and objects' attributes) that is required to implement Key Per I/O key injection functionality, see section 5.4 .

3.4 Communications

3.4.1 Communication Properties

The TPer SHALL support the minimum communication buffer size as defined in section 4.1.1.1. For each ComID, the physical buffer size SHALL be reported to the host via the `Properties` method.

The TPer SHALL terminate any IF-SEND command whose transfer length is greater than the reported MaxComPacketSize size for the corresponding ComID. For details, refer to "Invalid Transfer Length parameter on IF-SEND" in [5].

Data generated in response to methods contained within an IF-SEND command payload subpacket (including the required ComPacket / Packet / Subpacket overhead data) SHALL fit entirely within the response buffer. If the method response and its associated protocol overhead do not fit completely within the response buffer, the TPer

1) SHALL terminate processing of the IF-SEND command payload,

2) SHALL NOT return any part of the method response if the Sync Protocol is being used, and
3) SHALL return an empty response list with a TCG status code of RESPONSE_OVERFLOW in that method's response status list.

3.4.2 Supported Security Protocols

The TPer SHALL support:

- IF-RECV commands with a Security Protocol values of 0x00, 0x01, 0x02, and 0x03
- IF-SEND commands with a Security Protocol values of 0x01, 0x02.and 0x03

3.4.3 ComIDs

For the purpose of communication using Security Protocol 0x01, the TPer SHALL:

- support at least one statically allocated ComID for Synchronous Protocol communication;
- have the ComID Extension values = 0x0000 for all statically allocated ComIDs; and
- keep all statically allocated ComIDs in the Active state.

When the TPer receives an IF-SEND or IF-RECV with an inactive or unsupported ComID, the TPer SHALL either:

- terminate the command as defined in [5] with "Other Invalid Command Parameter", or
- follow the requirements defined in [3] for "IF-SEND to Inactive or Unsupported Reserved ComID" or "IF-RECV to Inactive or Unsupported Reserved ComID".

ComIDs SHALL be assigned based on the allocation presented in Table 23.

Table 23: ComID Assignments

ComID	Description
0x0000	Reserved
0x0001	Level 0 Device Discovery
0x0002-0x0003	Reserved for TCG
0x0004	TPER_RESET command
0x0005-0x07FF	Reserved for TCG
0x0800-0x0FFF	Vendor Unique
0x1000-0xFFFF	ComID management (Protocol ID=0x01 and 0x02)

3.4.4 Synchronous Protocol

The TPer SHALL support the Synchronous Protocol. The synchronous communications protocol uses configuration information defined by the values reported via Level 0 Discovery (see section 3.1.1), the host's communication properties, and a TPer's communication properties (see section 4.1.1.1). Refer to [3] for additional details.

3.4.4.1 Payload Encoding

3.4.4.1.1 Stream Encoding Modifications

The TPer SHALL support tokens listed in Table 24. If an unsupported token is encountered, the TPer SHALL treat the token as a streaming protocol violation and return an error per the definition in section 3.4.4.1.3.

Table 24: Supported Tokens

Token	Acronym
Tiny atom	N/A
Short atom	N/A
Medium atom	N/A
Long atom	N/A
Start List	SL
End List	EL
Start Name	SN
End Name	EN
Call	CALL
End of Data	EOD
End of session	EOS
Start transaction	ST
End of transaction	ET
Empty atom	MT

The TPer SHALL support the above token atoms with the B bit set to zero or one and the S bit set to zero.

3.4.4.1.2 TCG Packets

Within a single IF-SEND/IF-RECV command, the TPer SHALL support a ComPacket containing one Packet, which contains one Subpacket. The host may discover TPer support of capabilities beyond this requirement in the parameters returned in response to a `Properties` method.

The TPer MAY ignore Credit Control Subpackets sent by the host. The host may discover TPer support of Credit Management with Level 0 Discovery. For more details refer to Section 3.1.1 Level 0 Discovery (M)

The TPer MAY ignore the AckType and Acknowledgement fields in the Packet header on commands from the host and set these fields to zero in its responses to the host. The host may discover TPer support of the TCG packet acknowledgement/retry mechanism with Level 0 Discovery. For more details refer to Section 3.1.1 Level 0 Discovery (M)

The TPer MAY ignore packet sequence numbering and not enforce any sequencing behavior. Refer to [3] for details on discovery of packet sequence numbering support.

3.4.4.1.3 Payload Error Response

The TPer SHALL respond according to the following rules if it encounters a streaming protocol violation:

- If the error is on the Session Manager or is such that the TPer cannot resolve a valid session ID from the payload (i.e., errors in the ComPacket header or Packet header), then the TPer SHALL discard the payload and immediately transition to the “Awaiting IF-SEND” state.
- If the error occurs after the TPer has resolved the session ID, then the TPer SHALL abort the session and MAY prepare a `CloseSession` method for retrieval by the host.

3.4.5 Storage Device Resets

3.4.5.1 Interface Resets

Storage Device interface resets that generate TCG reset events are defined in [5].

Interface initiated TCG reset events SHALL result in:

1. All open sessions SHALL be aborted;
2. All uncommitted transactions SHALL be aborted;
3. All pending session startup activities SHALL be aborted;
4. All TCG command and response buffers SHALL be invalidated;
5. All related method processing SHALL be aborted;
6. For each ComID, the state of the synchronous protocol stack SHALL transition to “Awaiting IF-SEND” state; and
7. No notification of these events SHALL be sent to the host.

3.4.5.2 TCG Reset Events

Table 25 replaces the definition of TCG reset_types that are defined in [3]:

Table 25: reset_types

Enumeration value	Associated Value
0	Power Cycle
1	Hardware
2	HotPlug
3	Programmatic
4-15	Reserved
16-31	Vendor Unique

3.4.6 Protocol Stack Reset Commands (M)

An IF-SEND containing a Protocol Stack Reset Command SHALL be supported.

Refer to [3] for details.

4 Key Per I/O SSC-compliant Functions and SPs

4.1 Session Manager

4.1.1 Methods

4.1.1.1 Properties (M)

A Key Per I/O SSC compliant Storage Device SHALL support the `Properties` method. The requirements for support of the various TPer and Host properties, and the requirements for their values, are shown in Table 26 (see section 1.3.1 for the definitions of M and N)

Table 26: Properties Requirements

Property Name	TPer Property Requirements and Values Reported	Host Property Requirements and Values Accepted
MaxComPacketSize	(M) 2048 minimum	(M) Initial Assumption: 2048 Minimum allowed: 2048 Maximum allowed: VU
MaxResponseComPacketSize	(M) 2048 minimum	(N) Although this is a legal host property, there is no requirement for the TPer to use it. The TPer MAY ignore this host property and not list it in the HostProperties result of the `Properties` method response.
MaxPacketSize	(M) 2028 minimum	(M) Initial Assumption: 2028 Minimum allowed: 2028 Maximum allowed: VU
MaxIndTokenSize	(M) 1992 minimum	(M) Initial Assumption: 1992 Minimum allowed: 1992 Maximum allowed: VU
MaxPackets	(M) 1 minimum	(M) Initial Assumption: 1 Minimum allowed: 1 Maximum allowed: VU

Property Name	TPer Property Requirements and Values Reported	Host Property Requirements and Values Accepted
MaxSubpackets	(M) 1 minimum	(M) Initial Assumption: 1 Minimum allowed: 1 Maximum allowed: VU
MaxMethods	(M) 1 minimum	(M) Initial Assumption: 1 Minimum allowed: 1 Maximum allowed: VU
MaxSessions	(M) 1 minimum	N/A – not a host property
MaxAuthentications	(M) 2 minimum	N/A – not a host property
MaxTransactionLimit	(M) 1 minimum	N/A – not a host property
DefSessionTimeout	(M) VU	N/A – not a host property
Protocol3MaxPayloadSize	(M) 2048 minimum	(M) Initial Assumption: 2048 Minimum allowed: 2048 Maximum allowed: VU
Protocol3MaxKmipBatchItems	(M) 2 minimum	(M) Initial Assumption: 2 Minimum allowed: 2 Maximum allowed: VU

4.1.1.1.1 Protocol3MaxPayloadSize

If the Host sends a ComPacket with Security Protocol value of 0x03 whose size (including ComPacket header) exceeds the value of the TPer's Protocol3MaxPayloadSize property, a TPer SHALL abort the IF-SEND command as described in the "Invalid Transfer Length parameter on IF-SEND" section of the appropriate interface section of [4].

A TPer SHALL NOT send a ComPacket with Security Protocol value of 0x03 whose size (including ComPacket header) exceeds the value of the host's Protocol3MaxPayloadSize property.

4.1.1.1.2 Protocol3MaxKmipBatchItems

If the Host sends a ComPacket with Security Protocol value of 0x03 that contains more KMIP Batch Items than the value of the TPer's Protocol3MaxKmipBatchItems property, a TPer SHALL fail the KMIP Batch Items in the request message that is associated with the ComPacket with an appropriate failure result reason as specified in section 5.4.4.1.1.2.

A TPer SHALL NOT send a ComPacket with Security Protocol value of 0x03 that contains more KMIP Batch Items responses than the value of the Host's Protocol3MaxKmipBatchItems property.

4.1.1.2 StartSession (M)

A Key Per I/O SSC compliant Storage Device SHALL support the following parameters for the `StartSession` method:

- HostSessionID
- SPID
- Write = support for "True" is (M), support for "False" is (N)
- HostChallenge
- HostSigningAuthority

For a Key Per I/O SSC compliant Storage Device, a value of "True" for the Write parameter SHALL be supported.

For a Key Per I/O SSC compliant Storage Device, a value of "False" (i.e., read only session) for the Write parameter MAY be supported.

4.1.1.3 SyncSession (M)

A Key Per I/O SSC compliant Storage Device SHALL support the following parameters for the `SyncSession` method:

- HostSessionID
- SPSessionID

4.1.1.4 CloseSession (O)

A Key Per I/O SSC compliant Storage Device MAY support the `CloseSession` method.

4.2 Admin SP

The Admin SP includes the Base Template and the Admin Template.

4.2.1 Base Template Tables

All tables included in the following subsections are Mandatory.

4.2.1.1 SPInfo (M)

The SPInfo Table is defined in [3], and Table 27 defines the Preconfiguration Data for the SPInfo Table.

Table 27: Admin SP - SPInfo Table Preconfiguration

UID	SPID	Name	Size	SizeInUse	SPSessionTimeout	Enabled
00 00 00 02 00 00 00 01	00 00 02 05 00 00 00 01	"Admin"				T

4.2.1.2 SPTemplates (M)

The SPTemplates Table is defined in [3], and Table 28 defines the Preconfiguration Data for the SPTemplates Table.

*ST1 means this version number or any version number that complies with this SSC.

Table 28: Admin SP - SPTemplates Table Preconfiguration

UID	TemplateID	Name	Version
00 00 00 03 00 00 00 01	00 00 02 04 00 00 00 01	“Base”	00 00 00 02 *ST1
00 00 00 03 00 00 00 02	00 00 02 04 00 00 00 02	“Admin”	00 00 00 02 *ST1

4.2.1.3 Table (M)

The Table Table is defined in [3], and Table 29 defines the Preconfiguration Data for the Table Table.

Refer to section 5.3 for a description and requirements of the MandatoryWriteGranularity and RecommendedAccessGranularity columns.

Table 29: Admin SP - Table Table Preconfiguration

UID	Name	CommonName	TemplateID	Kind	Column	NumColumns	Rows	RowsFree	RowBytes	LastID	MinSize	MaxSize	MandatoryWrite Granularity	RecommendedAccess
00 00 00 01 00 00 00 01	“Table”			Object									0	0
00 00 00 01 00 00 00 02	“SPInfo”			Object									0	0
00 00 00 01 00 00 00 03	“SPTemplates”			Object									0	0
00 00 00 01 00 00 00 06	"MethodID"			Object									0	0
00 00 00 01 00 00 00 07	"AccessControl"			Object									0	0

UID	Name	CommonName	TemplateID	Kind	Column	NumColumns	Rows	RowsFree	RowBytes	LastID	MinSize	MaxSize	MandatoryWrite Granularity	RecommendedAccess
00 00 00 01 00 00 00 08	"ACE"			Object									0	0
00 00 00 01 00 00 00 09	"Authority"			Object									0	0
00 00 00 01 00 00 00 0B	"C_PIN"			Object									0	0
00 00 00 01 00 00 02 01	"TPerInfo"			Object									0	0
00 00 00 01 00 00 02 04	"Template"			Object									0	0
00 00 00 01 00 00 02 05	"SP"			Object									0	0
00 00 00 01 00 00 11 01	“DataRemovalMechanism”			Object									0	0

Start of informative comment

[3] states, “The `Table` Table in the Admin SP includes a row for each table that the TPer supports, in addition to a row for each table that exists in the Admin SP.” However, the Key Per I/O SSC requires only the tables from the Admin SP to be included in the Admin SP’s `Table` Table, as indicated in Table 29.

End of informative comment

4.2.1.4 MethodID (M)

The MethodID Table is defined in [3], and Table 30 defines the Preconfiguration Data for the MethodID Table.

*MT1: refer to section 5.1.2 for details on the requirements for supporting `Revert`.

*MT2: refer to section 5.1.1 for details on the requirements for supporting `Activate`.

.

Table 30: Admin SP - MethodID Table Preconfiguration

UID	Name	CommonName	TemplateID
00 00 00 06 00 00 00 08	"Next"		
00 00 00 06 00 00 00 0D	"GetACL"		
00 00 00 06 00 00 00 16	"Get"		
00 00 00 06 00 00 00 17	"Set"		
00 00 00 06 00 00 00 1C	"Authenticate"		
00 00 00 06 00 00 02 02 *MT1	"Revert"		
00 00 00 06 00 00 02 03 *MT2	"Activate"		
00 00 00 06 00 00 06 01	“Random”		

4.2.1.5 AccessControl (M)

Table 31 contains Optional rows identified by (O).

Notation:

*AC1: the notation of “TT TT TT TT” represents a shorthand for the LSBs of the Table object UIDs

*AC2: the notation of “TT TT TT TT” represents a shorthand for the LSBs of the SPTemplates object UIDs

*AC3: the notation of “TT TT TT TT” represents a shorthand for the LSBs of the MethodID object UIDs

*AC4: the notation of “TT TT TT TT” represents a shorthand for the LSBs of the ACE object UIDs

*AC5: the notation of “TT TT TT TT” represents a shorthand for the LSBs of the Authority object UIDs

*AC6: the notation of “TT TT TT TT” represents a shorthand for the LSBs of the Template object UIDs

*AC7: the notation of “TT TT TT TT” represents a shorthand for the LSBs of the SP object UIDs

Start of informative comment

*AC8: refer to section 5.1.2 for details on the requirements for supporting `Revert` method.

*AC9: refer to section 5.1.1 for details on the requirements for supporting `Activate` method.

End of informative comment

Notes:

- The InvokingID, MethodID and GetACLACL columns are a special case. Although they are marked as Read-Only with fixed access control, the access control for invocation of the `Get` method is (N).
- The ACL column is readable only via the `GetACL` method.

Table 31: Admin SP - AccessControl Table Preconfiguration

Table association - Informative text	UID	InvokingID	InvokingID Name - informative text	MethodID	CommonName	ACL	Log	AddACEACL	RemoveACEACL	GetACLACL	DeleteMethodACL	AddACELog	RemoveACELog	GetACLLog	DeleteMethodLog	LogTo
Table																
		00 00 00 01 00 00 00 00	Table	Next		ACE_Anybody				ACE_Anybody						
		00 00 00 01 TT TT TT TT *AC1	TableObj	Get		ACE_Anybody				ACE_Anybody						
SPInfo																
		00 00 00 02 00 00 00 01	SPInfoObj	Get		ACE_Anybody				ACE_Anybody						
SPTemplates																
		00 00 00 03 00 00 00 00	SPTemplates	Next		ACE_Anybody				ACE_Anybody						
		00 00 00 03 TT TT TT TT *AC2	SPTemplatesObj	Get		ACE_Anybody				ACE_Anybody						
MethodID																
		00 00 00 06 00 00 00 00	MethodID	Next		ACE_Anybody				ACE_Anybody						

Table association - Informative text	UID	InvokingID	InvokingID Name - informative text	MethodID	CommonName	ACL	Log	AddACEACL	RemoveACEACL	GetACLACL	DeleteMethodACL	AddACELog	RemoveACELog	GetACLLog	DeleteMethodLog	LogTo
		00 00 00 06 TT TT TT TT *AC3	MethodIDObj	Get		ACE_Anybody				ACE_Anybody						
ACE																
		00 00 00 08 00 00 00 00	ACE	Next		ACE_Anybody				ACE_Anybody						
		00 00 00 08 TT TT TT TT *AC4	ACEObj	Get		ACE_Anybody				ACE_Anybody						
Authority																
		00 00 00 09 00 00 00 00	Authority	Next		ACE_Anybody				ACE_Anybody						
		00 00 00 09 TT TT TT TT *AC5	AuthorityObj	Get		ACE_Anybody				ACE_Anybody						
		00 00 00 09 00 00 00 03	Makers	Set		ACE_Set_Enabled				ACE_Anybody						
		00 00 00 09 00 00 02 01	Admin1	Set		ACE_Set_Enabled				ACE_Anybody						

				C_PIN		Table association - Informative text
						UID
00 00 00 0B 00 00 84 02	00 00 00 0B 00 00 00 01	00 00 00 0B 00 00 00 01	00 00 00 0B 00 00 00 00		00 00 00 09 00 00 02 00 (+XX)	InvokingID
C_PIN_MSID	C_PIN_SID	C_PIN_SID	C_PIN		AdminXX	InvokingID Name - informative text
Get	Set	Get	Next		Set	MethodID
						CommonName
ACE_C_PIN_MSID_Get_PIN	ACE_C_PIN_SID_Set_PIN	ACE_C_PIN_SID_Get_NOPIN	ACE_Anybody		ACE_Set_Enabled	ACL
						Log
						AddACEACL
						RemoveACEACL
ACE_Anybody	ACE_Anybody	ACE_Anybody	ACE_Anybody		ACE_Anybody	GetACLACL
						DeleteMethodACL
						AddACELog
						RemoveACELog
						GetACLLog
						DeleteMethodLog
						LogTo

TPerInfo					**Table association - Informative text**
					UID
	00 00 00 0B 00 00 02 00 (+XX)	00 00 00 0B 00 00 02 01	00 00 00 0B 00 00 02 00 (+XX)	00 00 00 0B 00 00 02 01	**InvokingID**
	C_PIN_AdminXX	C_PIN_Admin1	C_PIN_AdminXX	C_PIN_Admin1	InvokingID Name - informative text
	Set	Set	Get	Get	**MethodID**
					CommonName
	ACE_C_PIN_Admins_Set_PIN	ACE_C_PIN_Admins_Set_PIN	ACE_C_PIN_SID_Get_NOPIN	ACE_C_PIN_SID_Get_NOPIN	**ACL**
					Log
					AddACEACL
					RemoveACEACL
	ACE_Anybody	ACE_Anybody	ACE_Anybody	ACE_Anybody	**GetACLACL**
					DeleteMethodACL
					AddACELog
					RemoveACELog
					GetACLLog
					DeleteMethodLog
					LogTo

		SP			Template			Table association - Informative text
								UID
00 00 00 00 00 00 00 01	00 00 00 00 00 00 00 01		00 00 02 04 TT TT TT TT *AC6	00 00 02 04 00 00 00 00		00 00 02 01 00 03 00 01	00 00 02 01 00 03 00 01	InvokingID
ThisSP	ThisSP		TemplateObj	Template		TPerInfoObj	TPerInfoObj	InvokingID Name - informative text
Random	Authenticate		Get	Next		Set	Get	MethodID
								CommonName
ACE_Anybody	ACE_Anybody		ACE_Anybody	ACE_Anybody		ACE_TPerInfo_Set_ProgrammaticResetEnable	ACE_Anybody	ACL
								Log
								AddACEACL
								RemoveACEACL
ACE_Anybody	ACE_Anybody		ACE_Anybody	ACE_Anybody		ACE_Anybody	ACE_Anybody	GetACLACL
								DeleteMethodACL
								AddACELog
								RemoveACELog
								GetACLLog
								DeleteMethodLog
								LogTo

Table association - Informative text	UID	InvokingID	InvokingID Name - informative text	MethodID	CommonName	ACL	Log	AddACEACL	RemoveACEACL	GetACLACL	DeleteMethodACL	AddACELog	RemoveACELog	GetACLLog	DeleteMethodLog	LogTo
		00 00 02 05 00 00 00 00	SP	Next		ACE_Anybody				ACE_Anybody						
		00 00 02 05 TT TT TT TT *AC7	SPObj	Get		ACE_Anybody				ACE_Anybody						
*AC8		00 00 02 05 TT TT TT TT *AC7	SPObj	Revert		ACE_SP_SID, ACE_Admin				ACE_Anybody						
*AC9		00 00 02 05 TT TT TT TT *AC7	SPObj	Activate		ACE_SP_SID				ACE_Anybody						
DataRemoval Mechanism																
		00 00 11 01 00 00 00 01	DataRemovalMechanismObj	Get		ACE_Anybody				ACE_Anybody						
		00 00 11 01 00 00 00 01	DataRemovalMechanismObj	Set		ACE_DataRemovalMechanism_Set				ACE_Anybody						

4.2.1.6 ACE (M)

Table 32 defines Admin SP's ACE Table preconfigured values.

Start of informative comment

*ACE1 means that row is (M) if the TPer supports either `Activate` or `Revert`, and (N) otherwise.

End of informative comment

Table 32: Admin SP - ACE Table Preconfiguration

Table Association - Informative text	UID	Name	CommonName	BooleanExpr	Columns
BaseACEs					
	00 00 00 08 00 00 00 01	"ACE_Anybody"		Anybody	All
	00 00 00 08 00 00 00 02	"ACE_Admin"		Admins	All
Authority					
	00 00 00 08 00 03 00 01	"ACE_Set_Enabled"		SID	Enabled
C_PIN					
	00 00 00 08 00 00 8C 02	"ACE_C_PIN_SID_Get_NOPIN"		Admins OR SID	UID, CharSet, TryLimit, Tries, Persistence
	00 00 00 08 00 00 8C 03	"ACE_C_PIN_SID_Set_PIN"		SID	PIN
	00 00 00 08 00 00 8C 04	"ACE_C_PIN_MSID_Get_PIN"		Anybody	UID, PIN
	00 00 00 08 00 03 A0 01	"ACE_C_PIN_Admins_Set_PIN"		Admins OR SID	PIN
TPerInfo					
	00 00 00 08 00 03 00 03	"ACE_TPerInfo_Set_ProgrammaticResetEnable"		SID	ProgrammaticResetEnable
SP					
*ACE1	00 00 00 08 00 03 00 02	"ACE_SP_SID"		SID	All
DataRemovalMechanism					
*ACE1	00 00 00 08 00 05 00 01	"ACE_DataRemovalMechanism_Set_ActiveDataRemovalMechanism"		Admins OR SID	ActiveDataRemovalMechanism

4.2.1.7 Authority (M)

The Authority Table is defined in [3], and Table 33 defines the Preconfiguration Data for the Authority Table.

Note:

- Admin1 (M) is required; any additional Admin authorities (O) are optional.

Table 33: Admin SP - Authority Table Preconfiguration

UID	Name	CommonName	IsClass	Class	Enabled	Secure	HashAndSign	PresentCertificat	Operation	Credential	ResponseSign	ResponseExch	ClockStart	ClockEnd	Limit	Uses	Log	LogTo
00 00 00 09 00 00 00 01	"Anybody"		F	Null	T	None	None	F	None	Null	Null	Null						
00 00 00 09 00 00 00 02	"Admins"		T	Null	T	None	None	F	None	Null	Null	Null						
00 00 00 09 00 00 00 03	"Makers"		T	Null	T	None	None	F	None	Null	Null	Null						
00 00 00 09 00 00 00 06	"SID"		F	Null	T	None	None	F	Password	C_PIN_SID	Null	Null						
00 00 00 09 00 00 02 01	"Admin1"		F	Admins	F	None	None	F	Password	C_PIN_Admin1	Null	Null						
00 00 00 09 00 00 02 00 (+XX)[1] (O)	"AdminXX"		F	Admins	F	None	None	F	Password	C_PIN_AdminXX	Null	Null						

4.2.1.8 C_PIN (M)

The C_PIN Table is defined in [3], and Table 34 defines the Preconfiguration Data for the C_PIN Table.

Table 34: Admin SP - C_PIN Table Preconfiguration

UID	Name	CommonName	PIN	CharSet	TryLimit	Tries	Persistence
00 00 00 0B 00 00 00 01	"C_PIN_SID"		VU	Null	**VU**	**VU**	FALSE
00 00 00 0B 00 00 84 02	"C_PIN_MSID"		**MSID**				
00 00 00 0B 00 00 02 01	"C_PIN_Admin1"		""	Null	**0**	**0**	FALSE
00 00 00 0B 00 00 02 00 (+XX) (O)	"C_PIN_AdminXX"		""	Null	**0**	**0**	FALSE

Start of informative comment

Several activation / take ownership models are possible. The simplest model is a process whereby the host discovers the initial C_PIN_SID PIN value by performing a `Get` operation on the C_PIN_MSID object. This model requires that the initial C_PIN_SID PIN be the value of the C_PIN_MSID PIN (see section 3.1.1.3.5).

Key Per I/O allows the initial C_PIN_SID PIN value to be vendor unique to allow for alternative activation / take ownership models. Such models require that the C_PIN_SID PIN be retrieved in a way that is beyond the scope of this specification.

Before a Storage Device's vendor chooses to implement an activation / take ownership model based on a vendor unique SID PIN, the vendor must undertake due diligence to ensure that the ecosystem exists to support such an activation / take ownership model. Having a C_PIN_SID PIN value that is different from the C_PIN_MSID PIN value may have serious ramifications, such as the inability to take ownership of the Storage Device.

See section 5.1.2 for an explanation of how `Revert` affects the value of the C_PIN_SID PIN column.

End of informative comment

4.2.2 Base Template Methods

Refer to section 4.2.1.4 for supported methods.

4.2.3 Admin Template Tables

4.2.3.1 TPerInfo (M)

The `TPerInfo` table has the column defined in Table 35, in addition to those defined in [3]:

Table 35: Admin SP – TPerInfo Columns

Column Number	Column Name	IsUnique	Colum Type
0x08	ProgrammaticResetEnable		boolean

- **ProgrammaticResetEnable**
  This column indicates whether support for programmatic resets is enabled or not. If ProgrammaticResetEnable is TRUE, then the TPER_RESET command is enabled. If ProgrammaticResetEnable is FALSE, then the TPER_RESET command is not enabled. This column is readable by Anybody and modifiable by the SID authority.

Table 36: Admin SP - TPerInfo Table Preconfiguration

UID	Bytes	GUDID	Generation	Firmware Version	ProtocolVersion	SpaceForIssuance	SSC	ProgrammaticResetEnable
00 00 02 01 00 03 00 01		**VU** *TP1			1 *TP2		["Key Per I/O"] *TP3	FALSE

Note:

- *TP1 means that the value in the GUDID column SHALL comply with the format defined in [3].
- *TP2 means that this version or any version that supports the defined features in this SSC.
- *TP3 means that the SSC column is a list of names and SHALL have "Key Per I/O" as one of the list elements.

4.2.3.2 Template (M)

The Template Table is defined in [3], and Table 37 defines the Preconfiguration Data for the Template Table.

Table 37: Admin SP - Template Table Preconfiguration

UID	Name	Revision Number	Instances	MaxInstances
00 00 02 04 00 00 00 01	"Base"	1	**VU**	**VU**
00 00 02 04 00 00 00 02	"Admin"	1	1	1

4.2.3.3 SP (M)

The SP Table is defined in [3], and Table 38 defines the Preconfiguration Data for the SP Table.

*SP1 means that this row only exists in the Admin SP's OFS when the Key Per I/O SP is created by the manufacturer.

Table 38: Admin SP - SP Table Preconfiguration

UID	Name	ORG	EffectiveAuth	DateOfIssue	Bytes	LifeCycle	Frozen
00 00 02 05 00 00 00 01	"Admin"					Manufactured	FALSE
00 00 02 05 00 00 00 03 *SP1	"KeyPerI/O"					Manufactured-Inactive	FALSE

4.2.4 Admin Template Methods

Refer to section 4.2.1.4 for supported methods.

4.2.5 Key Per I/O SSC Additional Column Types

4.2.5.1 Data_removal_mechanism

The data_removal_mechanism type is defined in Table 39 for Key Per I/O SSC:

Table 39: data_removal_mechanism Type Table Addition

UID	Name	Format
00 00 00 05 00 00 04 20	data_removal_mechanism	Enumeration_Type, 0, 7

Table 40 defines the enumeration values. The mechanisms associated with each Enumeration Value are defined in Table 7.

Table 40: data_removal_mechanism Enumeration Values

Enumeration Value	Associated Value
0	Overwrite Data Erase
1	Block Erase
2	Cryptographic Erase
3 – 4	Reserved
5	Vendor Specific Erase
6-7	Reserved

4.2.6 Key Per I/O SSC Additional Data Structures

4.2.6.1 DataRemovalMechanism (Object Table)

The `DataRemovalMechanism` table is defined in Table 41.

Table 41: DataRemovalMechanism Table Description

Column Number	Column Name	IsUnique	Column Type
0x00	UID		uid
0x01	ActiveDataRemovalMechanism		data_removal_mechanism

4.2.6.1.1 UID

This is the unique identifier of this row in the `DataRemovalMechanism` table.

This column SHALL NOT be modifiable by the host.

4.2.6.1.2 ActiveDataRemovalMechanism

This column value selects which Data Removal Mechanism in the Supported Data Removal Mechanism field in the Supported Data Removal Mechanism feature descriptor is active and will be used to remove data upon execution of the `Revert` method. If an attempt is made to set the ActiveDataRemovalMechanism column value to an unsupported value of the data_removal_mechanism type, then the `Set` method invocation SHALL result in the method failing with the status INVALID_PARAMETER.

4.2.7 Key Per I/O SSC Additional Tables

4.2.7.1 DataRemovalMechanism (M)

The `DataRemovalMechanism` table SHALL contain exactly one row with `UID=0x00 0x00 0x11 0x01 0x00 0x00 0x00 0x01`. The `DataRemovalMechanism` table SHALL be supported (see Table 42).

Table 42: Admin SP – DataRemovalMechanism Table Preconfiguration

UID	ActiveDataRemovalMechanism
00 00 11 01 00 00 00 01	VU

4.2.8 Crypto Template Tables

A Key Per I/O SSC compliant Storage Device is not required to support any Crypto template tables.

4.2.9 Crypto Template Methods

Refer to section 4.2.1.4 for supported methods.

4.2.9.1 Random

The TPer SHALL implement the `Random` method with the constraints stated in this subsection. TPer support of the following parameter is Mandatory:

- `Count`

The TPer SHALL support `Count` values less than or equal to 32.

Attempts to use unsupported parameters SHALL result in a method failure response with TCG status INVALID_PARAMETER.

4.3 Key Per I/O SP

4.3.1 Base Template Tables

All tables defined with (M) in section titles are Mandatory.

4.3.1.1 SPInfo (M)

The SPInfo Table is defined in [3], and Table 43 defines the Preconfiguration Data for the SPInfo Table.

Table 43: Key Per I/O SP - SPInfo Table Preconfiguration

UID	SPID	Name	Size	SizeInUse	SPSessionTimeout	Enabled
00 00 00 02 00 00 00 01	00 00 02 05 00 00 00 03	"KeyPerI/O"				T

4.3.1.2 SPTemplates (M)

The SPTemplates Table is defined in [3], and Table 44 defines the Preconfiguration Data for the SPTemplates Table.

*SP1 means that this version number or any number that supports the defined features in this SSC

Table 44: Key Per I/O SP - SPTemplates Table Preconfiguration

UID	TemplateID	Name	Version
00 00 00 03 00 00 00 01	00 00 02 04 00 00 00 01	"Base"	00 00 00 02 *SP1

4.3.1.3 Table (M)

The Table Table is defined in [3], and Table 45 defines the Preconfiguration Data for the Table Table.

Table 45 contains Optional rows designated with (O).

Table 45: Key Per I/O SP - Table Table Preconfiguration

UID	Name	CommonName	TemplateID	Kind	Column	NumColumns	Rows	RowsFree	RowBytes	LastID	MinSize	MaxSize	MandatoryWrite Granularity	RecommendedAccess Granularity
00 00 00 01 00 00 00 01	"Table"			Object									0	0
00 00 00 01 00 00 00 02	"SPInfo"			Object									0	0
00 00 00 01 00 00 00 03	"SPTemplates"			Object									0	0
00 00 00 01 00 00 00 06	"MethodID"			Object									0	0

UID	Name	CommonName	TemplateID	Kind	Column	NumColumns	Rows	RowsFree	RowBytes	LastID	MinSize	MaxSize	MandatoryWrite Granularity	RecommendedAccess Granularity
00 00 00 01 00 00 00 07	"AccessControl"			Object									0	0
00 00 00 01 00 00 00 08	"ACE"			Object									0	0
00 00 00 01 00 00 00 09	"Authority"			Object									0	0
00 00 00 01 00 00 00 0B	"C_PIN"			Object									0	0
00 00 00 01 00 00 00 0A (O)	“Certificates”			Object									0	0
00 00 00 01 00 00 12 01	“KeyTagAllocation”			Object									0	0
00 00 00 01 00 00 12 02	“KeyEncryptionKey”			Object									0	0
00 00 00 01 00 00 12 03	“KPIOPolicies”			Object									0	0
00 00 00 01 00 00 12 04 (O)	“KeyPerIOPublicKeyCertificateData”			Byte									VU	VU
00 00 00 01 00 00 10 01	“DataStore”			Byte									VU	VU

4.3.1.4 Type (N)

The `Type` table is not required (N) by Key Per I/O SSC. The following types as defined by [3] SHALL meet the following requirements:

- The "boolean_ACE" type (00000005 0000040E) SHALL include the OR Boolean operator.
- To accommodate all possible Admin Authority OR combinations (e.g., Admin1 OR Admin2 OR…AdminN), the "AC_element" type (00000005 00000801) SHALL support at least 2*N-1 entries, where N is the number of supported Admin authorities (N Admin authorities, N-1 Boolean operators).

4.3.1.4.1 New kek_obj_uidref type

A Key Per I/O SSC compliant Storage Device SHALL support the kek_obj_uidref type as specified in Table 46:

Table 46: kek_obj_uidref type

UID	Name	Format
00 00 00 05 00 00 0C 1F	kek_obj_uidref	Restricted_Reference_Type{6}, uidref {KeyEncryptionKeyTableUID}

4.3.1.4.2 New kek_obj_uidref_list type

A Key Per I/O SSC compliant Storage Device SSC SHALL support the kek_obj_uidref_list type as specified in Table 47. See [3] for the definition of the asterisk.

Table 47: kek_obj_uidref_list type

UID	Name	Format
00 00 00 05 00 00 08 08	kek_obj_uidref_list	List_Type, * kek_obj_uidref

4.3.1.4.3 New KeyEncryptionKey_Type type

A Key Per I/O SSC compliant Storage Device SSC SHALL support the KeyEncryptionKey_Type type as specified in Table 48.

Table 48: KeyEncryptionKey_Type

UID	Name	Format
00 00 00 05 00 00 06 08	KeyEncryptionKey_Type	Alternative_Type, bytes_16, bytes_32

4.3.1.5 MethodID (M)

The MethodID Table is defined in [3], and Table 49 defines the Preconfiguration Data for the MethodID Table.

Table 49: Key Per I/O SP - MethodID Table Preconfiguration

UID	Name	CommonName	TemplateID
00 00 00 06 00 00 00 08	"Next"		
00 00 00 06 00 00 00 0D	"GetACL"		
00 00 00 06 00 00 00 16	"Get"		
00 00 00 06 00 00 00 17	"Set"		
00 00 00 06 00 00 00 1C	"Authenticate"		
00 00 00 06 00 00 06 01	“Random”		

4.3.1.6 AccessControl (M)

Table 50's InvokingID column contains an Optional row designated with (O).

*AC1: the notation of "TT TT TT TT" represents a shorthand for the LSBs of the Table object UIDs

*AC2: the notation of "TT TT TT TT" represents a shorthand for the LSBs of the SPTemplates object UIDs

*AC3: the notation of "TT TT TT TT" represents a shorthand for the LSBs of the MethodID object UIDs

*AC4: the notation of "TT TT TT TT" represents a shorthand for the LSBs of the ACE object UIDs

*AC5: the notation of "TT TT TT TT" represents a shorthand for the LSB of the Authority object UIDs

Notes:

- The `AccessControl` Table is different from any other table defined in this specification. Although cells in this table are marked as Read-Only with fixed access control, the access control for invocation of the `Get` method is (N).
- The ACL column is readable only via the `GetACL` method.

Table 50: Key Per I/O SP - AccessControl Table Preconfiguration

Table Association - informative only	UID	InvokingID	InvokingID Name - informative only	MethodID	CommonName	ACL	Log	AddACEACL	RemoveACEACL	GetACLACL	DeleteMethodACL	AddACELog	RemoveACELog	GetACLLog	DeleteMethodLog	LogTo
SP																
		00 00 00 00 00 00 00 01	ThisSP	Authenticate		ACE_Anybody				ACE_Anybody						
		00 00 00 00 00 00 00 01	ThisSP	Random		ACE_Anybody				ACE_Anybody						
Table																
		00 00 00 01 00 00 00 00	Table	Next		ACE_Anybody				ACE_Anybody						
		00 00 00 01 TT TT TT TT *AC1	TableObj	Get		ACE_Anybody				ACE_Anybody						
SPInfo																

Table Association - informative only	UID	InvokingID	InvokingID Name - informative only	MethodID	CommonName	ACL	Log	AddACEACL	RemoveACEACL	GetACLACL	DeleteMethodACL	AddACELog	RemoveACELog	GetACLLog	DeleteMethodLog	LogTo
		00 00 00 02 00 00 00 01	SPInfoObj	Get		ACE_Anybody				ACE_Anybody						
SPTemplates																
		00 00 00 03 00 00 00 00	SPTemplates	Next		ACE_Anybody				ACE_Anybody						
		00 00 00 03 TT TT TT TT *AC2	SPTemplatesObj	Get		ACE_Anybody				ACE_Anybody						
MethodID																
		00 00 00 06 00 00 00 00	MethodID	Next		ACE_Anybody				ACE_Anybody						
		00 00 00 06 TT TT TT TT *AC3	MethodIDObj	Get		ACE_Anybody				ACE_Anybody						
ACE																
		00 00 00 08 00 00 00 00	ACE	Next		ACE_Anybody				ACE_Anybody						
		00 00 00 08 TT TT TT TT *AC4	ACEObj	Get		ACE_ACE_Get_All				ACE_Anybody						

Table Association - informative only																
	UID	**InvokingID**	InvokingID Name - informative only	**MethodID**	**CommonName**	**ACL**	**Log**	**AddACEACL**	**RemoveACEACL**	**GetACLACL**	**DeleteMethodACL**	**AddACELog**	**RemoveACELog**	**GetACLLog**	**DeleteMethodLog**	**LogTo**
		00 00 00 08 00 03 80 00	ACE_ACE_Get_All	Set		ACE_ACE_Set_BooleanExpression				ACE_Anybody						
		00 00 00 08 00 03 90 00	ACE_Authority_Get_All	Set		ACE_ACE_Set_BooleanExpression				ACE_Anybody						
		00 00 00 08 00 03 FC 00	ACE_DataStore_Get_All	Set		ACE_ACE_Set_BooleanExpression				ACE_Anybody						

Table Association - informative only	UID	InvokingID	InvokingID Name - informative only	MethodID	CommonName	ACL	Log	AddACEACL	RemoveACEACL	GetACLACL	DeleteMethodACL	AddACELog	RemoveACELog	GetACLLog	DeleteMethodLog	LogTo
		00 00 00 08 00 03 FC 01	ACE_DataStore_Set_All	Set		ACE_ACE_Set_BooleanExpression				ACE_Anybody						
Authority																
		00 00 00 09 00 00 00 00	Authority	Next		ACE_Anybody				ACE_Anybody						
		00 00 00 09 TT TT TT TT *AC5	AuthorityObj	Get		ACE_Authority_Get_All, ACE_Anybody_Get_CommonName				ACE_Anybody						

		C_PIN			Table Association - informative only
					UID
00 00 00 0B 00 01 00 01	00 00 00 0B 00 00 00 00		00 00 00 09 00 01 00 00 (+XX XX) (O)	00 00 00 09 00 01 00 01	**InvokingID**
C_PIN_Admin1	C_PIN		AdminXXXX	Admin1	InvokingID Name - informative only
Get	Next		Set	Set	**MethodID**
					CommonName
ACE_C_PIN_Admins_Get_All_NOPIN	ACE_Anybody		ACE_Authority_Set_Enabled, ACE_Admins_Set_CommonName	ACE_Admins_Set_CommonName	**ACL**
					Log
					AddACEACL
					RemoveACEACL
ACE_Anybody	ACE_Anybody		ACE_Anybody	ACE_Anybody	**GetACLACL**
					DeleteMethodACL
					AddACELog
					RemoveACELog
					GetACLLog
					DeleteMethodLog
					LogTo

	Certificates				Table Association - informative only
					UID
00 00 00 0A 00 00 00 00		00 00 00 0B 00 01 00 00 (+XX XX)	00 00 00 0B 00 01 00 01	00 00 00 0B 00 01 00 00 (+ XX XX)	**InvokingID**
Certificates		C_PIN_AdminXXXX	C_PIN_Admin1	C_PIN_AdminXXXX	InvokingID Name - informative only
Next		Set	Set	Get	**MethodID**
					CommonName
ACE_Anybody		ACE_C_PIN_Admins_Set_PIN	ACE_C_PIN_Admins_Set_PIN	ACE_C_PIN_Admins_Get_All_NOPIN	**ACL**
					Log
					AddACEACL
					RemoveACEACL
		ACE_Anybody	ACE_Anybody	ACE_Anybody	**GetACLACL**
					DeleteMethodACL
					AddACELog
					RemoveACELog
					GetACLLog
					DeleteMethodLog
					LogTo

Table Association - informative only	UID	InvokingID	InvokingID Name - informative only	MethodID	CommonName	ACL	Log	AddACEACL	RemoveACEACL	GetACLACL	DeleteMethodACL	AddACELog	RemoveACELog	GetACLLog	DeleteMethodLog	LogTo
		00 00 00 0A 00 00 00 01	Certificate1	Get		ACE_Anybody				ACE_Anybody						
KeyTagAllocation																
		00 00 12 01 00 00 00 00	KeyTagAllocation	Next		ACE_Anybody				ACE_Anybody						
		00 00 12 01 00 00 00 00 (+XXXX)	KeyTagAllocationXXXX	Get		ACE_KeyTagAllocation_Admin_ Get				ACE_Anybody						
		00 00 12 01 00 00 00 00 (+XXXX)	KeyTagAllocationXXXX	Set		ACE_KeyTagAllocation_Admin_ Set				ACE_Anybody						
KeyEncryptionKey																

				Table Association - informative only
				UID
00 00 12 02 00 01 00 00 (+XXXX)	00 00 12 02 00 00 00 02	00 00 12 02 00 00 00 01	00 00 12 02 00 00 00 00	**InvokingID**
KeyEncryptionKeyXXXX	PKIPublicKeyEncryptionKey	NULLKeyEncryptionKey	KeyEncryptionKey	InvokingID Name - informative only
Get	Get	Get	Next	**MethodID**
				CommonName
ACE_KeyEncryptionKey_Admin_Get	ACE_KeyEncryptionKey_Admin_Get	ACE_KeyEncryptionKey_Admin_Get	ACE_Anybody	**ACL**
				Log
				AddACEACL
				RemoveACEACL
ACE_Anybody	ACE_Anybody	ACE_Anybody	ACE_Anybody	**GetACLACL**
				DeleteMethodACL
				AddACELog
				RemoveACELog
				GetACLLog
				DeleteMethodLog
				LogTo

Table Association - informative only	UID	InvokingID	InvokingID Name - informative only	MethodID	CommonName	ACL	Log	AddACEACL	RemoveACEACL	GetACLACL	DeleteMethodACL	AddACELog	RemoveACELog	GetACLLog	DeleteMethodLog	LogTo
		00 00 12 02 00 01 00 00 (+XXXX)	KeyEncryptionKeyXXXX	Set		ACE_KeyEncryptionKey_Admin_Set				ACE_Anybody						
KPIOPolicies																
		00 00 12 03 00 00 00 01	KPIOPoliciesObj	Get		ACE_KPIOPolicies_Admin_Get				ACE_Anybody						
		00 00 12 03 00 00 00 01	KPIOPoliciesObj	Set		ACE_KPIOPolicies_Admin_Set				ACE_Anybody						
KeyPerIOPublicKeyCertificateData																

Table Association - informative only	UID	InvokingID	InvokingID Name - informative only	MethodID	CommonName	ACL	Log	AddACEACL	RemoveACEACL	GetACLACL	DeleteMethodACL	AddACELog	RemoveACELog	GetACLLog	DeleteMethodLog	LogTo
		00 00 12 04 00 00 00 00	KeyPerIOPublicKeyCertificateData	Get		ACE_Anybody				ACE_Anybody						
DataStore																
		00 00 10 01 00 00 00 00	DataStore	Get		ACE_DataStore_Get_All				ACE_Anybody						
		00 00 10 01 00 00 00 00	DataStore	Set		ACE_DataStore_Set_All				ACE_Anybody						

4.3.1.7 ACE (M)

Table 51 defines Access Control Elements (ACEs) for Key Per I/O SP Base Template. The TPer SHALL fail the `Set` method invocation with status INVALID_PARAMETER if the host attempts to set a value not supported by the TPer.

Table 51: Key Per I/O SP - ACE Table Preconfiguration

Table Association -Informative Column	UID	Name	CommonName	BooleanExpr	Columns
Base ACEs					
	00 00 00 08 00 00 00 01	"ACE_Anybody"		Anybody	All
	00 00 00 08 00 00 00 02	"ACE_Admin"		Admins	All
	00 00 00 08 00 00 00 03	"ACE_Anybody_Get_CommonName"		Anybody	UID, CommonName

Table Association -Informative Column	UID	Name	CommonName	BooleanExpr	Columns
	00 00 00 08 00 00 00 04	"ACE_Admins_Set_CommonName"		Admins	CommonName
ACE					
	00 00 00 08 00 03 80 00	"ACE_ACE_Get_All"		Admins	All
	00 00 00 08 00 03 80 01	"ACE_ACE_Set_BooleanExpression"		Admins	BooleanExpr
Authority					
	00 00 00 08 00 03 90 00	"ACE_Authority_Get_All"		Admins	All
	00 00 00 08 00 03 90 01	"ACE_Authority_Set_Enabled"		Admins	Enabled
C_PIN					
	00 00 00 08 00 03 A0 00	"ACE_C_PIN_Admins_Get_All_NOPIN"		Admins	UID, CharSet, TryLimit, Tries, Persistence
	00 00 00 08 00 03 A0 01	"ACE_C_PIN_Admins_Set_PIN"		Admins	PIN
KPIOPolicies					
	00 00 00 08 00 06 00 01	“ACE_KPIOPolicies_Admin_Set”		Admins	ClearSingleMEKAllowed, ClearAllMEKsAllowed, ReplayProtectionEnabled, PlaintextKEKProgrammingEnabled, PKIProtectedKEKProgrammingEnabled, KeyInjectionInterfaceLockEnabled, KeyInjectionInterfaceLocked, KeyInjectionInterfaceLockOnReset
	00 00 00 08 00 06 00 02	“ACE_KPIOPolicies_Admin_Get”		Admins	All
KeyTagAllocation					
	00 00 00 08 00 06 10 01	“ACE_KeyTagAllocation_Admin_Set”		Admins	Managed, NumberOfKeyTags, AllowedKeyEncryptionKeys
	00 00 00 08 00 06 10 02	“ACE_KeyTagAllocation_Admin_Get”		Admins	All
KeyEncryptionKey					

Table Association -Informative Column	UID	Name	CommonName	BooleanExpr	Columns
	00 00 00 08 00 06 20 01	"ACE_KeyEncryptionKey_Admin_Set"		Admins	AccessLockEnabled, AccessLocked, LockOnReset, AllowedKeyEncryptionKeys
	00 00 00 08 00 06 20 02	"ACE_KeyEncryptionKey_Admin_Get"		Admins	UID, AccessLockEnabled, AccessLocked, LockOnReset, AllowedKeyEncryptionKeys
DataStore					
	00 00 00 08 00 03 FC 00	"ACE_DataStore_Get_All"		Admins	All
	00 00 00 08 00 03 FC 01	"ACE_DataStore_Set_All"		Admins	All

4.3.1.8 Authority (M)

Table 52's UID column contains an Optional row designated with (O).

Notes:

1. Admin1 is required; any additional Admin authorities are (O).

Table 52: Key Per I/O SP - Authority Table Preconfiguration

UID	Name	CommonName	IsClass	Class	Enabled	Secure	HashAndSign	PresentCertificate	Operation	Credential	ResponseSign	ResponseExch	ClockStart	ClockEnd	Limit	Uses	Log	LogTo
00 00 00 09 00 00 00 01	"Anybody"	""	F	Null	T	None	None	F	None	Null	Null	Null						
00 00 00 09 00 00 00 02	"Admins"	""	T	Null	T	None	None	F	None	Null	Null	Null						

UID	Name	CommonName	IsClass	Class	Enabled	Secure	HashAndSign	PresentCertificate	Operation	Credential	ResponseSign	ResponseExch	ClockStart	ClockEnd	Limit	Uses	Log	LogTo
00 00 00 09 00 01 00 01	"Admin1"	""	F	Admins	T	None	None	F	Password	C_PIN_Admin1	Null	Null						
00 00 00 09 00 01 00 00 (+XX XX)[1] (O)	"AdminXXXX"	""	F	Admins	F	None	None	F	Password	C_PIN_AdminXXXX	Null	Null						

4.3.1.9 C_PIN (M)

Table 53's UID column includes an Optional row designated with (O).

Notes:

1. If the Key Per I/O SP's original life cycle state is Manufactured-Inactive, see 5.1.1.2 for the initial value of C_PIN_Admin1.PIN. If the Key Per I/O SP's original life cycle state is Manufactured, then the initial value of C_PIN_Admin1.PIN is the same as the Admin SP's C_PIN_MSID.PIN value.

Table 53: Key Per I/O SP - C_PIN Table Preconfiguration

UID	Name	CommonName	PIN	CharSet	TryLimit	Tries	Persistence
00 00 00 0B 00 01 00 01	"C_PIN_Admin1"		SID or MSID[1]	Null	0	0	FALSE
00 00 00 0B 00 01 00 00 (+XX XX) (O)	"C_PIN_AdminXXXX"		""	Null	0	0	FALSE

4.3.1.10 Certificates (O)

The `Certificates` Table is defined in [3]. Table 54 shows the Preconfiguration data for the `Certificates` Table.

The UID in the CertData column refers to the `KeyPerIOPublicKeyCertificateData` byte table (see 4.3.5.4) that holds the certificate data for the Certificate1 object.

A Key Per I/O compliant Storage Device that supports PKI-based KEK Transport SHALL support this `Certificates` table.

Table 54: Key Per I/O SP - Certificates Table Preconfiguration

UID	Name	CommonName	CertData	CertSize
00 00 00 0A 00 00 00 01	"Certificate1"		00 00 12 04 00 00 00 00	VU

4.3.2 Base Template Methods

Refer to section 4.3.1.5 for supported methods.

4.3.3 Crypto Template Tables

A Key Per I/O SSC compliant Storage Device is not required to support any Crypto template tables.

4.3.4 Crypto Template Methods

Refer to section 4.3.1.5 for supported methods.

4.3.4.1 Random

Refer to section 4.2.9.1 for additional constraints imposed on the `Random` method.

4.3.5 SSC Specific Tables

4.3.5.1 KPIOPolicies (M)

The `KPIOPolicies` Table allows the host to configure Key Per I/O policies that apply to the entire Key Per I/O TPer instance. The host can discover which policies it has enabled at any time by querying the table using the `Get` method.

Table 55: KPIOPolicies Table Definition

Column Number	Column Name	IsUnique	Column Type
0x00	UID		uid
0x01	ClearSingleMEKAllowed		boolean
0x02	ClearAllMEKsAllowed		boolean
0x03	ReplayProtectionEnabled		boolean
0x04	PKIProtectedKEKProgrammingEnabled		boolean
0x05	PlaintextKEKProgrammingEnabled		boolean
0x06	KeyInjectionInterfaceLockEnabled		boolean
0x07	KeyInjectionInterfaceLocked		boolean
0x08	KeyInjectionInterfaceLockOnReset		reset_types

4.3.5.1.1 UID

The value of this column is a unique identifier of the row of the `KPIOPolicies` Table.

The UID column SHALL NOT be modifiable by the host.

4.3.5.1.2 ClearSingleMEKAllowed

The value of this column determines whether the ClearSingleMEK command is enabled or disabled. The command SHALL be enabled when the ClearSingleMEKAllowed is True. The command SHALL be disabled when the ClearSingleMEKAllowed is False.

4.3.5.1.3 ClearAllMEKsAllowed

The value of this column determines whether the ClearAllMEKs command is enabled or disabled. The command SHALL be enabled when the ClearAllMEKsAllowed is True. The command SHALL be disabled when the ClearAllMEKsAllowed is False.

4.3.5.1.4 ReplayProtectionEnabled

The value of this column determines whether all symmetric key injections require replay protected payloads (as defined in section 3.2.6) or not.

If the value of this column is set to True,

a) the Get Nonce command SHALL be enabled and processed as specified in section 3.2.6;
b) the Level 0 Replay Protection Enabled bit field SHALL be set to one; and
c) the TPer SHALL ensure that every wrapped key injected includes a Nonce value generated by the Get Nonce command as specified by sections 5.4.4.2.3 or 5.4.4.2.4.

If the value of the ReplayProtectionEnabled column is set to True and the injected wrapped key (i.e., KEK or MEK) does not include a Nonce value, then the TPer SHALL fail the key injection operation that failed to include the Nonce value with Invalid Message Result Reason.

Start of informative comment

A Nonce value can only be used once per KMIP Request Message when replay protection is enabled on the Storage Device. For a KMIP Request Message that may contain multiple Batch Items, a Storage Device may allow reuse of the same Nonce value for each Batch Item within that message since individual Batch Items cannot be replayed to the Storage Device outside of a KMIP Request Message context.

End of informative comment

A Storage Device SHALL NOT allow a Nonce value to be reused across multiple KMIP Request Messages. A Nonce value MAY be reused across multiple Batch Items within the same KMIP Request Message.

If a Storage Device unwraps the encrypted key, extracts the Nonce value (see section 5.4.4.2.2.3), and detects that the provided Nonce has already been used by a previous KMIP Request Message, then the Storage Device SHALL fail the key injection operation that included it with Cryptographic Failure Result Reason. The process by which a Storage Device determines that the Nonce has already been used by a previous KMIP Request Message is outside the scope of this specification.

If the value of this column is set to False,

a) the Get Nonce command SHALL be disabled as specified in section 3.2.6;
b) the Level 0 Replay Protection Enabled bit field SHALL be cleared to zero; and
c) the TPer SHALL ignore the Nonce value extracted from the wrapped key value and process the rest of the key injection operation normally (see sections 5.4.4.2.3 and 5.4.4.2.4 for details on key extraction processing).

4.3.5.1.5 PlaintextKEKProgrammingEnabled

Start of informative comment

By setting the PlaintextKEKProgrammingEnabled policy to True, the host turns on the TPer's processing of plaintext KEKs provisioning for any row of the KeyEncryptionKey Table that is associated with a KEK on Storage Devices that support the Plaintext KEK Provisioning option. Setting PlaintextKEKProgrammingEnabled to True allows any row of the KeyEncryptionKey Table that is associated with a KEK to accept a plaintext KEK. Setting PlaintextKEKProgrammingEnabled to True does not prohibit the injection of an encrypted KEK to any row of the KeyEncryptionKey Table.

By setting the PlaintextKEKProgrammingEnabled policy to False, the host allows the AllowedKeyEncryptionKeys column of the KeyEncryptionKey Table to exclusively determine how each individual KEK object is provisioned.

Attempts to inject a plaintext KEK in a specific row of the KeyEncryptionKey Table while this policy is disabled will be rejected by the TPer if that row is not configured to allow plaintext KEK (see section 4.3.5.3.10 for details).

End of informative comment

The value of the PlaintextKEKProgrammingEnabled column determines whether the TPer is allowed to successfully process KEKs injected in plaintext form.

If the value of the PlaintextKEKProgrammingEnabled column is set to True, the Storage Device SHALL accept plaintext KEKs for the TCG KEK Obj UID specified by the KEK injection request regardless of whether or not the NULLKeyEncryptionKey UID is included in the AllowedKeyEncryptionKeys list for the specified TCG KEK Obj UID.

If the value of the PlaintextKEKProgrammingEnabled column is set to True and the Storage Device receives an encrypted KEK, the Storage Device SHALL consult the KeyEncryptionKey Table's AllowedKeyEncryptionKeys column for the specified TCG KEK Obj UID to validate that the key that was used to wrap the injected KEK is allowed to be used for wrapping a KEK targeted to KeyEncryptionKey Table row associated with the TCG KEK Obj UID specified by the KEK injection request.

If the value of the PlaintextKEKProgrammingEnabled column is set to False, the Storage Device SHALL consult the KeyEncryptionKey Table's AllowedKeyEncryptionKeys column to determine how to process the KEK injected into a KeyEncryptionKey Table row associated with the TCG KEK Obj UID specified by the KEK injection request (see section 4.3.5.3.7 for additional details).

A TPer SHALL set the PlaintextKEKProgrammingEnabled column to False if the TPer does not support Plaintext KEK Provisioning as indicated by the Plaintext KEK Provisioning Supported bit of the Level 0 data structure.

Invoking a Set method on PlaintextKEKProgrammingEnabled column SHALL result in failure with a status of INVALID_PARAMETER if Plaintext KEK Provisioning is not supported.

4.3.5.1.6 PKIProtectedKEKProgrammingEnabled

Start of informative comment

By setting the PKIProtectedKEKProgrammingEnabled policy to True, the host turns on the TPer's processing of injected KEKs protected using RSA-OAEP for any row of the KeyEncryptionKey Table that is associated with KEK on Storage Devices that support the PKI-based KEK Transport option. Setting PKIProtectedKEKProgrammingEnabled to True allows any row of the KeyEncryptionKey Table that is associated with a KEK to accept an RSA-OAEP protected KEK. This setting does not prohibit the injection of KEKs protected using other algorithms.

By setting the PKIProtectedKEKProgrammingEnabled policy to False, the host allows the AllowedKeyEncryptionKeys column of the KeyEncryptionKey Table to exclusively determine how each individual KEK object is provisioned.

Attempts to inject a KEK protected using RSA-OAEP in a specific row of the KeyEncryptionKey Table while this policy is disabled will be rejected by the TPer if that row is not configured to allow RSA-OAEP protected KEKs (see section 4.3.5.3.10 for details).

End of informative comment

The value of the PKIProtectedKEKProgrammingEnabled column determines whether the TPer is allowed to successfully process injected KEKs protected via RSA-OAEP using a Storage Device's Key Per I/O Public Key certificate.

If the value of the PKIProtectedKEKProgrammingEnabled column is set to True and the name of the Certificate1 object is used as the Unique Identifier of the key that was used to protect the injected KEK, then the Storage Device SHALL retrieve the private key associated with the Storage Device's Key Per I/O Public Key Certificate and use it to unwrap the injected KEK and load the unwrapped KEK in the KeyEncryptionKey Table row associated with the TCG Obj UID specified by the KEK injection request, regardless of whether or not the PKIPublicKeyEncryptionKey UID is included in the allowed KEKs list for the specified TCG KEK Obj UID (see section 4.3.5.3.7 for details on allowed KEKs list interactions with KEK Object UID).

If the value of the PKIProtectedKEKProgrammingEnabled column is set to True and the name of the Certificate1 object is not used as the Unique Identifier of the key that was used to protect the injected KEK, then the Storage Device SHALL consult the KeyEncryptionKey Table's AllowedKeyEncryptionKeys column for the specified TCG KEK Obj UID to validate that the key that was used to wrap the injected KEK is allowed to be used for wrapping a KEK targeted to KeyEncryptionKey Table row associated with the TCG KEK Obj UID specified by the KEK injection request.

If the value of the PKIProtectedKEKProgrammingEnabled column is set to False, the Storage Device SHALL consult the KeyEncryptionKey Table's AllowedKeyEncryptionKeys column to determine how to process the KEK injected into a KeyEncryptionKey Table row associated with the TCG KEK Obj UID specified by the KEK injection request (see section 4.3.5.3.7) for additional details).

A TPer SHALL set the PKIProtectedKEKProgrammingEnabled column to False if the TPer does not support PKI-based KEK Transport as indicated by the PKI-based KEK Transport bit of the Level 0 data structure.

Invoking a Set method on PKIProtectedKEKProgrammingEnabled column SHALL result in failure with a status of INVALID_PARAMETER if PKI-based KEK Transport is not supported.

4.3.5.1.7 KeyInjectionInterfaceLockEnabled

The value of the KeyInjectionInterfaceLockEnabled column determines whether the KMIP Key Injection Interface locking feature (see 5.4.4.2.1) is enabled or not and whether the KeyInjectionInterfaceLocked column value is meaningful for a specific policy object in the KPIOPolicies Table.

If the value of the KeyInjectionInterfaceLockEnabled column is set to False, the KMIP Key Injection Interface locking feature SHALL be disabled, and the value of the KeyInjectionInterfaceLocked column SHALL be ignored.

If the value of the KeyInjectionInterfaceLockEnabled column is set to True, the KMIP Key Injection Interface locking feature SHALL be enabled, and the value of the KeyInjectionInterfaceLocked column SHALL determine the current lock state of KMIP key injection interface.

4.3.5.1.8 KeyInjectionInterfaceLocked

The value of the KeyInjectionInterfaceLocked column determines the lock status of the KMIP Key Injection Interface if the KeyInjectionInterfaceLockEnabled column is True.

If the value of the KeyInjectionInterfaceLocked column is set to True, then the KMIP Key Injection Interface SHALL be disabled. If the value of this column is set to False, then the KMIP Key Injection Interface SHALL be enabled.

The value of the KeyInjectionInterfaceLocked column SHALL be ignored if the KeyInjectionInterfaceLockEnabled column is set to False.

The `Set` method may be invoked by the host to change the value of this column and alter the KMIP Key Injection Interface lock state if the value of the KeyInjectionInterfaceLockEnabled column is set to True.

4.3.5.1.9 KeyInjectionInterfaceLockOnReset

The value of the KeyInjectionInterfaceLockOnReset column defines the TCG reset types that cause the value of the KeyInjectionInterfaceLocked column to be set to True if the KeyInjectionInterfaceLockEnabled column is set to True.

4.3.5.1.10 KPIOPolicies Table Preconfiguration

Table 56 defines the Preconfiguration Data for the `KPIOPolicies` Table.

*KP1 = Only a limited set of KeyInjectionInterfaceLockOnReset values is required to be supported by Key Per I/O SSC Storage Devices. Refer to section 4.3.5.3.11 for details.

Table 56: KPIOPolicies Table Preconfiguration

UID	ClearSingleMEKAllowed	ClearAllMEKsAllowed	ReplayProtectionEnabled	PlaintextKEKProgrammingEnabled	PKIProtectedKEKProgrammingEnabled	KeyInjectionInterfaceLockEnabled	KeyInjectionInterfaceLocked	KeyInjectionInterfaceLockOnReset
00 00 12 03 00 00 00 01	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	Power Cycle *KP1

4.3.5.1.11 KeyInjectionInterfaceLockOnReset Restrictions

The TPer SHALL support the following KeyInjectionInterfaceLockOnReset 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 KeyInjectionInterfaceLockOnReset 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)

4.3.5.2 KeyTagAllocation (M)

The `KeyTagAllocation` table allows the host to allocate a certain number of Key Tags to specific Namespaces. In addition, multiple Key Encryption Keys can be configured per Namespace via the AllowedKeyEncryptionKeys column.

The `KeyTagAllocation` Table is defined in Table 57.

Table 57: KeyTagAllocation Table Definition

Column Number	Column Name	IsUnique	Column Type
0x00	UID		uid
0x01	Name	Yes	name
0x02	CommonName	Yes	name
0x03	NamespaceID		bytes_4
0x04	Managed		boolean
0x05	NumberOfKeyTags		uinteger_2
0x06	AllowedKeyEncryptionKeys		kek_obj_uidref_list

4.3.5.2.1 UID

The value of this column is unique identifier of the row of the `KeyTagAllocation` table.

The UID column SHALL NOT be modifiable by the host.

4.3.5.2.2 Name

This is the name of the KeyTagAllocation object.

The Name column SHALL NOT be modifiable by the host.

4.3.5.2.3 CommonName

This is a name that MAY be shared among multiple KeyTagAllocation object.

4.3.5.2.4 NamespaceID

The NamespaceID column is the Namespace ID for the Namespace that is associated with a specific `KeyTagAllocation` Table row.

The NamespaceID SHALL NOT be modifiable by the host.

4.3.5.2.5 Managed

The Managed column determines whether the Namespace associated with the `KeyTagAllocation` Table row is managed by the Key Per I/O SP.

If the value of the Managed column is set to 1, then the Key Per I/O SP SHALL manage the associated Namespace and if the value of the Managed column is set to 0, then the Key Per I/O SP SHALL NOT manage the associated Namespace.

If the `Set` method is successfully invoked on a row in the `KeyTagAllocation` table that transitions the Managed column value from 0 to 1, then the method SHALL cause all existing user data to no longer be recoverable for the

affected Namespace. The mechanism for how the Storage Device makes user data not recoverable is implementation specific.
Because transitioning the Managed column value from 0 to 1 enables Key Per I/O capability for this Namespace, the Key Per I/O Enabled in Namespace (KPIOENS) bit of the Identify Namespace Data Structure (see [15]) SHALL be set to communicate this transition to the host (see section 5.11 for details). Additionally, the Namespace Level 0 descriptor SHALL also be updated to reflect this change (see section 3.1.2.5.4 for details).

If the `Set` method is successfully invoked on a row in the `KeyTagAllocation` table that transitions the Managed column value from 1 to 0, then the method SHALL clear all Media Encryption Keys for the Namespace Key Tags associated with the specified Namespace ID from the TPer's key cache. This action SHALL transition the specified NamespaceID out of the management of the Key Per I/O SP. As a result, the Key Per I/O Enabled in Namespace (KPIOENS) bit of the Identify Namespace Data Structure SHALL be cleared to zero to indicate that the Key Per I/O capability is disabled on this Namespace.

If a Namespace is not managed by the Key Per I/O SP, then the Clear Single MEK and Clear All MEKs commands SHALL fail with an appropriate error when the associated NamespaceID is specified for those commands. See sections 3.2.4 and 3.2.5 for more details.

If the Key Per I/O Scope field of the Key Per I/O's Feature Descriptor is set to one (see section 3.1.1.3.9), and the `Set` method is invoked on any row in the `KeyTagAllocation` Table with the Managed column specified, then the `Set` method SHALL fail with a status of INVALID_PARAMETER.

4.3.5.2.6 NumberOfKeyTags

The NumberOfKeyTags column is the number of Key Tags that are associated with a specific `KeyTagAllocation` Table row.

As section 2.2.5 specifies, the TPer SHALL support a Key Tag value starting at 0 for each Namespace managed by Key Per I/O SP and the Key Tag value SHALL be incremented by 1 for each associated Key Tag.

Start of informative comment

For example, if the column value of the NumberOfKeyTags column is set to 2, the supported Key Tags values will be Key Tag 0 and Key Tag 1.

End of informative comment

If the value of the NumberOfKeyTags column is cleared to 0 for a specific `KeyTagAllocation` Table row, Clear Single MEK SHALL fail when the associated NamespaceID is specified for those commands.

A Key Per I/O compliant TPer SHALL ensure that the value of the NumberOfKeyTags column for a specific `KeyTagAllocation` Table row does not exceed the value reported by the Maximum Number of Key Tags Supported Per Namespace field of the Level0 Discovery data (see 3.1.1.3.27). In addition, the TPer SHALL ensure that the total number of Key Tags allocated across all `KeyTagAllocation` Table rows does not exceed the value reported by the Total Number of Key Tags Supported field of the Level0 Discovery data (see 3.1.1.3.26).

Invoking a `Set` method on a specific `KeyTagAllocation` Table row with a value of the NumberOfKeyTags column that exceeds the Maximum Number of Key Tags Supported Per Namespace as reported in Level 0 Discovery SHALL result in a failure with a status of INVALID_PARAMETER (see [3]).

In addition, invoking a `Set` method on a specific `KeyTagAllocation` Table row with a value of the NumberOfKeyTags that would result in the sum of all entries of the NumberOfKeyTags column values across all `KeyTagAllocation` Table rows exceeding the total number of Key Tags supported by the Storage Device SHALL result in failure with a status of INVALID_PARAMETER (see [3]).

If the Set method is invoked on a specific KeyTagAllocation Table row:

a. with a value for the NumberOfKeyTags column set to a value that is less than the current column value of the NumberOfKeyTags column; and
b. the Set method would otherwise complete successfully,

then, the Storage Device SHALL decrease the number of Key Tags associated with the indicated KeyTagAllocation Table row, starting from the higher-indexed Key Tags.

Start of informative comment

For example, if the NumberOfKeyTags is reduced from three to one, then the TPer is expected to remove Key Tag 2 and Key Tag 1 and keep only Key Tag 0.

End of informative comment

Care should be taken to ensure that no Media Encryption Key is associated with a Key Tag that is being removed. The TPer SHALL fail a Set method invocation that attempts to remove Key Tags associated with Media Encryption Keys with a status of NOT_AUTHORIZED (see [3]).

Invoking a Set method on the NumberOfKeyTags column on the row in the KeyTagAllocation Table associated with a Namespace that is not managed by Key Per I/O SP SHALL result in failure with a status of INVALID_PARAMETER (see [3]).

4.3.5.2.7 AllowedKeyEncryptionKeys

The AllowedKeyEncryptionKeys column SHALL be the list of UIDs representing the KeyEncryptionKey objects from the KeyEncryptionKey Table that are associated with a specific KeyTagAllocation Table row.

For an MEK injection to a specific Namespace managed by Key Per I/O SP to be successful, the MEK is expected to be wrapped using a KEK currently associated with the KeyTagAllocation Table row that is associated with the Namespace.

The UIDs associated with PKIPublicKeyEncryptionKey row and the NULLKeyEncryptionKey row of the KeyEncryptionKey Table are not permitted to be associated with any KeyTagAllocation Table row. If an attempt is made to set the AllowedKeyEncryptionKeys column value in the KeyTagAllocation Table row associated with a Namespace managed by Key Per I/O SP to include PKIPublicKeyEncryptionKey UID or the NULLKeyEncryptionKey UID, then the Set method invocation SHALL result in failure with a status of INVALID_PARAMETER.

Invoking a Set method on AllowedKeyEncryptionKeys column on the row in the KeyTagAllocation Table associated with a Namespace that is not managed by Key Per I/O SP SHALL result in failure with a status of INVALID_PARAMETER.

4.3.5.2.8 KeyTagAllocation Table Preconfiguration

The following table contains Optional rows designated with (O).

Table 58 defines the Preconfiguration Data for the KeyTagAllocation Table. See [3] for the definition of the empty bracket.

KTA1 = The minimum number of required KeyTagAllocation Table rows is equal to the number of Namespaces supported by the TPer.

KTA2 = The preconfigured value of the Managed column for all rows of the `KeyTagAllocation` Table SHALL be set to one if the value reported by Key Per I/O Scope field of the Key Per I/O's Feature Descriptor is one. Otherwise, the preconfigured value of the Managed column for all rows SHALL be cleared to zero.

KAT3 = The value of the NumberOfKeyTags column SHALL be at least 1 for each NamespaceID that is managed by Key Per I/O SP.

Table 58: KeyTagAllocation Table Preconfiguration

UID	Name	CommonName	NamespaceID	Managed	NumberOfKeyTags	AllowedKeyEncryptionKeys
00 00 12 01 00 00 00 01	"KeyTagAllocation1"		1	VU *KTA2	VU *KTA3	[ ]
00 00 12 01 00 00 00 00 (+XX XX) (O)	"KeyTagAllocationXXXX" *KTA1		XXXX	VU *KTA2	VU *KTA3	[ ]

4.3.5.3 KeyEncryptionKey (M)

The `KeyEncryptionKey` Table allows the host to insert a Key Encryption Key to protect subsequent Media Encryption Keys injection or other Key Encryption Key Injections. The `KeyEncryptionKey` Table is defined in Table 59.

Table 59: KeyEncryptionKey Table Definition

Column Number	Column Name	IsUnique	Column Type
0x00	UID		uid
0x01	Name	Yes	name
0x02	CommonName	Yes	name
0x03	AccessLockEnabled		boolean
0x04	AccessLocked		boolean
0x05	LockOnReset		reset_types

Column Number	Column Name	IsUnique	Column Type
0x06	AllowedKeyEncryptionKeys		kek_obj_uidref_list
0x07	KMIPKeyUID		
0x08	Key		KeyEncryptionKey_Type

4.3.5.3.1 UID
The value of this column is a unique identifier of the row of the `KeyEncryptionKey` Table.

The UID column SHALL NOT be modifiable by the host.

4.3.5.3.2 Name
This is the name of the KeyEncryptionKey object.

The Name column SHALL NOT be modifiable by the host.

4.3.5.3.3 CommonName
This is a name that may be shared among multiple KeyEncryptionKey objects.

4.3.5.3.4 AccessLockEnabled
The value of this column determines whether or not the locking feature is enabled for a specific KeyEncryptionKey object in the `KeyEncryptionKey` Table and determines whether or not the AccessLocked column value is meaningful for a specific KeyEncryptionKey object in the `KeyEncryptionKey` Table.

4.3.5.3.5 AccessLocked
Start of informative comment

An AccessLocked column value indicates whether the usage of a KEK associated with a `KeyEncryptionKey` Table row is permitted or disallowed. When the AccessLocked column is set to True for a particular `KeyEncryptionKey` Table row, the KEK associated with that row can't be modified or be used to unwrap other keys (KEKs or MEKs). Setting AccessLocked column to True does not affect any key that was successfully injected protected using the key associated with the `KeyEncryptionKey` Table row whose AccessLocked column is set to True. A TPer does not track the relationship between a KEK and the keys wrapped by that KEK beyond a successful unwrap.

End of informative comment

The value of this column SHALL be the current access lock state for a specific KeyEncryptionKey object in the `KeyEncryptionKey` Table.

If the value of the AccessLockEnabled column is True and the value of the AccessLocked column is True for a specific KeyEncryptionKey object in the `KeyEncryptionKey` Table, then use of the Key that has been set in the Key column for that KeyEncryptionKey object SHALL be blocked. See sections 5.4.4.2.3 and 5.4.4.2.4 for the appropriate error statuses returned if the injected key is wrapped with a key in access locked state or if the key being updated is in access locked state.

A TPer SHALL NOT block the usage of any key previously injected protected by a KEK associated with a `KeyEncryptionKey` Table row whose AccessLocked column is set to True.

4.3.5.3.6 LockOnReset
The value of the LockOnReset column defines the TCG reset types that cause the value of the AccessLocked column to be set to True if the AccessLockEnabled column value is equal to True for the KeyEncryptionKey object.

4.3.5.3.7 AllowedKeyEncryptionKeys

The AllowedKeyEncryptionKeys column SHALL be the list of UIDs representing the KeyEncryptionKey objects from the `KeyEncryptionKey` Table that are allowed to be used to update the KEK associated with a specific `KeyEncryptionKey` Table row.

For a KEK injection to a specific row of the `KeyEncryptionKey` Table to be successful, the KEK is expected to be wrapped using a KEK currently listed in the AllowedKeyEncryptionKeys column for that row. An exception exists if the PlaintextKEKProgrammingEnabled or PKIProtectedKEKProgrammingEnabled policies are supported (see sections 4.3.5.1.5 and 4.3.5.1.6 for details of these policies).

If an attempt is made to set the AllowedKeyEncryptionKeys column value in the `KeyEncryptionKey` Table row associated with a specific TCG KEK Obj UID to include PKIPublicKeyEncryptionKey UID on a Storage Device that does not support PKI-based KEK Transport option, then the `Set` method invocation SHALL result in failure with a status of INVALID_PARAMETER.

If an attempt is made to set the AllowedKeyEncryptionKeys column value in the `KeyEncryptionKey` Table row associated with a specific TCG KEK Obj UID to include NULLKeyEncryptionKey UID on a Storage Device that does not support Plaintext KEK Provisioning option, then the `Set` method invocation SHALL result in failure with a status of INVALID_PARAMETER.

4.3.5.3.8 KMIP KeyUID

The value of this column SHALL be the KMIP unique identifier that is used to reference the key associated with the KeyEncryptionKey object.

4.3.5.3.9 Key

The value of this column identifies a key that is associated with the KeyEncryptionKey object. When a key is associated with a KeyEncryptionKey object, that key SHALL be used by the TPer for operations that identify the key using the value of the KeyUID column for the KeyEncryptionKey object.

Invoking `Get` method on this column SHALL fail with a status of NOT_AUTHORIZED.

If the value of the Key column is empty, the TPer SHALL allow a plaintext KEK to be provisioned for this key regardless of the configuration of the AllowedKeyEncryptionKeys column.

4.3.5.3.10 KeyEncryptionKey Table Preconfiguration

The following table contains Optional rows designated with (O).

*KEK1 = The usage of NULLKeyEncryptionKey UID enables or disables provisioning of a plaintext KEK for each `KeyEncryptionKey` Table row when the PlaintextKEKProgrammingEnabled policy is disabled. If the UID for the NULLKeyEncryptionKey is included in the AllowedKeyEncryptionKeys column for a specific KeyEncryptionKey object, then the TPer SHALL accept an unwrapped KEK to be updated for this KeyEncryptionKey object.

As stated in section 2.2.3.1.1, a Storage Device that supports Plaintext KEK Provisioning option SHALL support the NULLKeyEncryptionKey UID.

*KEK2 = Only a limited set of LockOnReset values are required to be supported by Key Per I/O SSC Storage Devices. Refer to section 4.3.5.3.11 for details.

*KEK3 = The usage of PKIPublicKeyEncryptionKey UID enables or disables the usage of RSA-OAEP encryption per each `KeyEncryptionKey` Table row when the PKIProtectedKEKProgrammingEnabled policy is disabled. If the UID for the PKIPublicKeyEncryptionKey is included in the AllowedKeyEncryptionKeys column for a specific

KeyEncryptionKey object, then the TPer SHALL accept a KEK wrapped with RSA-OAEP for that KeyEncryptionKey object.

As stated in section 2.2.3.2, A Storage Device that supports PKI-based KEK Transport option SHALL support the PKIPublicKeyEncryptionKey UID.

*KEK4 = KeyEncryptionKey1 is Mandatory. All other KeyEncryptionKey objects are Optional. A TPer SHOULD support at least one KeyEncryptionKey per supported Namespace.

Table 60: KeyEncryptionKey Table Preconfiguration

UID	Name	CommonName	AccessLockEnabled	AccessLocked	LockOnReset	AllowedKeyEncryptionKeys	KMIPKeyUID	Key
00 00 12 02 00 00 00 01 (O)	“NULLKeyEncryptionKey” *KEK1							
00 00 12 02 00 00 00 02 (O)	“PKIPublicKeyEncryptionKey” *KEK3							
00 00 12 02 00 01 00 01	“KeyEncryptionKey1"		F	F	Power Cycle *KEK2	KeyEncryption Key1	KMIP UID	
00 00 12 02 00 01 00 00 (+XX XX) (O) *KEK4	“KeyEncryptionKeyXXXX"		F	F	Power Cycle *KEK2	KeyEncryption KeyXXXX		

Start of informative comment

The default configurations of the Key column and the AllowedKeyEncryptionKeys column as shown in Table 60 allow for optimized provisioning flows where all KEK injections can be accomplished through a key injection protocol like KMIP without having to make changes to the `KeyEncryptionKey` table. The `KeyEncryptionKey` Table's Key column defaulting to empty (i.e., no key) allows for the first provisioning of KEKs to be done with a plaintext KEK. Because AllowedKeyEncryptionKeys' column default is that the only allowed KEK for updating subsequent KEK is the current KEK, a new KEK can then be provisioned wrapped with the old KEK without having to make changes to the `KeyEncryptionKey` Table.

If the host were to instead specify a `KeyEncryptionKey` Table row UID that implies a plaintext key (i.e., NULLKeyEncryptionKey UID) and include that UID in the AllowedKeyEncryptionKeys for the targeted row of the `KeyEncryptionKey` Table, disallowing plaintext KEK provisioning after first provisioning for that row would require the host to make changes to the `KeyEncryptionKey` Table.

End of informative comment

4.3.5.3.11 LockOnReset Restrictions

A TPer SHALL support the following LockOnReset 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 LockOnReset 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)

4.3.5.4 KeyPerIOPublicKeyCertificateData (O)

If supported by a TPer, the `KeyPerIOPublicKeyCertificateData` table allows the host to discover an X.509 Key Per I/O Public Key Certificate provisioned by the Storage Device manufacturer into the Storage Device prior to Key Per I/O SP activation. The public key contained in the certificate is used by the host for securing keys when they are injected into the Storage Device (see section 2.2.3.2).

A Key Per I/O compliant Storage Device that supports PKI-based KEK Transport SHALL support the `KeyPerIOPublicKeyCertificateData` Table.

Start of informative comment

The specifics of generating, provisioning, rotating, or revoking this certificate are outside the scope of this specification. It is strongly recommended, however, that the pre-provisioned certificate is unique per Storage Device and is generated using a cryptographically approved key generator and a random seeds generator as recommended by NIST SP800-56B (see [13]) and NIST FIPS186-4 (see [14]) specifications.

End of informative comment

The `KeyPerIOPublicKeyCertificateData` Table is a byte table.

The contents of the `KeyPerIOPublicKeyCertificateData` Table SHALL be in X.509 binary DER-encoded format (see [18]).

The contents of the `KeyPerIOPublicKeyCertificateData` Table SHALL be vendor unique.

The host is not allowed to make modifications to the contents of the table.

Anybody can retrieve the contents of the table.

5 Appendix – Key Per I/O SSC Specific Features

5.1 Key Per I/O SSC-Specific Methods

5.1.1 Activate – Admin Template SP Object Method

`Activate` is a Key Per I/O SSC-specific method for managing the life cycle of SPs created in manufacturing (Manufactured SP), whose initial life cycle state is "Manufactured-Inactive". The following pseudo-code is the signature of the `Activate` Method (see [3] for more information).

```
SPObjectUID.Activate[ ]
=>
[ ]
```

`Activate` is an object method that operates on objects in the Admin SP's `SP` table. A TPer SHALL NOT permit `Activate` to be invoked on the SP objects of issued SPs.

Invocation of `Activate` on an SP object that is in the "Manufactured-Inactive" state causes the SP to transition to the "Manufactured" state. Invocation of `Activate` on an SP in any other life cycle state SHALL complete successfully provided access control is satisfied and have no effect. The `Activate` method allows the TPer owner to "turn on" an SP that was created in manufacturing.

This method operates within a Read-Write session to the Admin SP. The SP SHALL be activated immediately after the method returns success if its invocation is not contained within a transaction.

In case of an "Activate Error" (see [5]) `Activate` SHALL fail with a status of FAIL.

The MethodID for `Activate` SHALL be 00 00 00 06 00 00 02 03.

5.1.1.1 Activate Support

Support for `Activate` within transactions is (N), and the behavior is out of the scope of this document.

Support for `Activate` on the Key Per I/O SP's object in the `SP` Table is Mandatory.

5.1.1.2 Side effects of Activate

Upon successful activation of an SP that was in the "Manufactured-Inactive" state, the following changes SHALL be made:

- The LifeCycleState column of SP's object in the Admin SP's `SP` table SHALL change to "Manufactured".
- The current SID PIN (C_PIN_SID) in the Admin SP is copied into the `PIN` column of Admin1's C_PIN credential (C_PIN_Admin1) in the activated SP. This allows for taking ownership of the SP with a known PIN credential.
- Any TPer functionality affected by the life cycle state of the SP based on the templates incorporated into it is modified as defined in the appropriate Template reference section of [3], and as defined in section 5.2.2.2 and section 5.2.2.3 of this specification.

Start of informative comment

When a Storage Device's controller indicates that the Key Per I/O capability applies to all Namespaces in the NVM subsystem (i.e., the Key Per I/O Scope (KPIOSC) bit is set to one (see [15])), activating the Key Per I/O SP transitions the Storage Device from a state where it generates and manages all media encryption keys to a state where the host manages all media encryption keys on all Namespaces. This will lead to the loss of all previous data on all Namespaces, so as to not erode the security of the media encryption keys that were under the Storage Device's management.

The implication of this is that Key Per I/O SP activation with this KPIOSC configuration will not be compatible with activation models where OSes and other management software may need to be installed prior to activating Key Per I/O SP. Such models can instead benefit from having a bootstrapping Namespace that isn't

managed by Key Per I/O (i.e., the KPIOSC bit is cleared to zero) at activation time whose data will remain accessible to the host through Key Per I/O SP activation.

End of informative comment

- Unless already in the Manufactured life cycle state, activating the Key Per I/O SP SHALL cause all existing user data to no longer be accessible if the Key Per I/O Scope bit of the Key Per I/O Capabilities field in the Identify Controller data structure is set to one (see section 5.12). The mechanism for how a Storage Device makes data inaccessible is implementation specific.
  If the Key Per I/O capability does not apply to all Namespaces in the NVM subsystem and is independently enabled and disabled on each Namespace within the NVM subsystem (i.e., Bit 1 of the Key Per I/O Capabilities field in the Identify Controller data structure is cleared to zero), activating the Key Per I/O SP does not cause existing data to no longer be accessible (see section 5.12), because the independent enablement and disablement of the Key Per I/O capability on each Namespace can only take place once the Key Per I/O SP has successfully been transitioned into Manufactured life cycle state (see section 4.3.5.2.5 for details).

5.1.2 Revert – Admin Template SP Object Method

`Revert` is a Key Per I/O SSC-specific method for managing the life cycle of SPs created in manufacturing (Manufactured SP). The following pseudo-code is the signature of the `Revert` Method (see [3] for more information).

```
SPObjectUID.Revert[ ]
=>
[ ]
```

`Revert` is an object method that operates on objects in the Admin SP's `SP` table. A TPer SHALL NOT permit `Revert` to be invoked on the SP objects of issued SPs.

Invoking `Revert` on an SP object causes the SP to revert to its Original Factory State. This method allows the TPer owner (or TPer manufacturer, if access control permits and the Maker authorities are enabled) to remove the SP owner's ownership of the SP and revert the SP to its Original Factory State.

Invocation of `Revert` is permitted on Manufactured SPs that are in any life cycle state. Successful invocation of `Revert` on a Manufactured SP that is in the Manufactured-Inactive life cycle state SHALL have no effect on the SP.

This method operates within a Read-Write session to the Admin SP. The TPer SHALL revert the SP immediately after the method is successfully invoked outside of a transaction. If the `Revert` method is invoked on the Admin SP's object in the SP table, the TPer SHALL abort the session immediately after reporting status of the method invocation if invoked outside of a transaction. The TPer MAY prepare a `CloseSession` method for retrieval by the host to indicate that the session has been aborted.

The MethodID for `Revert` SHALL be 00 00 00 06 00 00 02 02.

5.1.2.1 Revert Support

Support for `Revert` within transactions is (N), and the behavior is outside the scope of this document.

Support for `Revert` on the Admin SP's object in the `SP` table is mandatory. (Note that the OFS of the Admin SP is Manufactured, see section 5.2.2).

Support for `Revert` on the Key Per I/O SP's object in the `SP` Table is mandatory.

5.1.2.2 Side effects of Revert

Upon successful invocation of the `Revert` method, the following changes SHALL be made:

- If the Key Per I/O SP is not in the Manufactured-Inactive life cycle state, then successful invocation of the `Revert` method on the Key Per I/O SP or Admin SP SHALL cause user data removal as defined by the

ActiveDataRemovalMechanism (see Table 40) on all Namespaces managed by Key Per I/O SP. Namespaces which are not managed by Key Per I/O SP SHALL not be impacted.

- If the Key Per I/O SP is in the Manufactured-Inactive life cycle state, then successful invocation of the `Revert` method on the Key Per I/O SP SHALL NOT cause user data removal in the Storage Device.

Interactions with interface commands during the processing of the `Revert` method are defined in [5].

If any TCG reset occurs prior to completing user data removal in the Storage Device, then the operation SHALL be aborted and the Key Per I/O SP SHALL NOT revert to its Original Factory State.

Start of informative comment

If any TCG reset occurs during the processing of the `Revert` method, the result of user data removal is undefined and the TPer does not erase personalization of the Key Per I/O SP. For example, the PIN column value for each row in C_PIN table is unchanged.

End of informative comment

Upon completion of user data removal in the Storage Device, or if the Key Per I/O SP is in the Manufactured-Inactive life cycle state, the following changes SHALL be made:

- The row in the Admin SP's `SP` table that represents this SP SHALL revert to its original factory values.
- The SP itself SHALL revert to its Original Factory State. While reverting to its Original Factory State, a TPer SHALL securely erase all personalization of the SP and revert the personalized values to their original factory values. The mechanism for secure erasure is implementation specific.
- When the `Revert` method is successfully invoked on the SP object for the Admin SP (UID = 00 00 02 05 00 00 00 01), the ***entire TPer*** SHALL revert to its Original Factory State, including:
  - All personalization of the Admin SP itself, except for the PIN column value of the C_PIN_SID object. See section 5.1.2.2.1 for the effects of `Revert` upon the PIN column value of the C_PIN_SID object.
  - All issued SPs SHALL be deleted, and all Manufactured SPs SHALL revert to Original Factory State. Manufactured SPs that were in the Manufactured-Inactive life cycle state SHALL be unaffected.
- Any TPer functionality affected by the life cycle state of the SP based on the templates incorporated into it is modified as defined in the appropriate Template reference section of [3], and as defined in section 5.2.2.2 and section 5.2.2.3 of this specification.

5.1.2.2.1 Effects of Revert on the PIN Column Value of C_PIN_SID

When the `Revert` method is successfully invoked on the SP object for the Admin SP (UID = 00 00 02 05 00 00 00 01), the PIN column value of the C_PIN_SID object SHALL be affected as follows:

1. If the SID authority has never been successfully authenticated, then the C_PIN_SID PIN column SHALL remain at its current value.
2. If the SID authority has previously been successfully authenticated, then:
   a) If the value of the "Behavior of C_PIN_SID PIN upon TPer Revert" field in the Key Per I/O SSC V1.00 Level 0 Feature Descriptor is 0x00, then the C_PIN_SID PIN column SHALL be set to the PIN column value of the C_PIN_MSID object. Additionally, the "Initial C_PIN_SID PIN Indicator" field SHALL be set to 0x00 upon completion of the `Revert`.
   b) If the value of the "Behavior of C_PIN_SID PIN upon TPer Revert" field in the Key Per I/O SSC V2.00 Level 0 Feature Descriptor is not 0x00, then the C_PIN_SID PIN column SHALL be set to a vendor unique (VU) value.

5.2 Life Cycle

5.2.1 Issued vs Manufactured SPs

5.2.1.1 Issued SPs

A Key Per I/O SSC compliant Storage Device SHALL NOT support issuance (see [3] for the definition of Issuance).

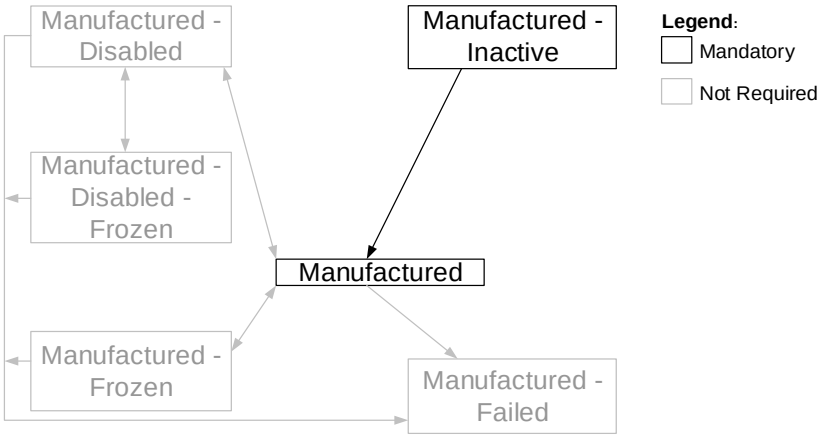
5.2.1.2 Manufactured SPs

Key Per I/O SSC-compliant SPs that are created in manufacturing (Manufactured SPs) SHALL NOT have an implementation-specific life cycle and SHALL conform to the life cycle defined in section 5.2.2.

5.2.2 Manufactured SP Life Cycle States

The state diagram for Manufactured SPs is shown in Figure 11.

Figure 11: Life Cycle State Diagram for Manufactured SPs



Additional state transitions may exist, depending on the states supported by the Storage Device and the SP's Original Factory State. Invoking `Revert` (see section 5.1.2) on the SP will cause the SP to transition back to its Original Factory State.

The Original Factory State of the Admin SP SHALL be Manufactured. The only state that is mandatory for the Admin SP is Manufactured.

The Key Per I/O SP's Original Factory State SHALL be Manufactured-Inactive.

The Key Per I/O SP SHALL support the Manufactured-Inactive and Manufactured states.

The other states in Figure 11 are beyond the scope of this document.

5.2.2.1 State definitions for Manufactured SPs

1. **Manufactured-Inactive**: This is the Original Factory State for SPs that are created in manufacturing, where it is not desirable for the functionality of that SP to be active when the TPer is shipped. All templates that exist in an SP that is in the Manufactured-Inactive state SHALL be counted in the `Instances` column of the appropriate objects in the Admin SP's `Template` table. Sessions cannot be opened to SPs in the Manufactured-Inactive state. Only SPs whose Original Factory State was Manufactured-Inactive can return to the Manufactured-Inactive state.

   Support for the Manufactured-Inactive state is Mandatory for the Key Per I/O SP.

2. **Manufactured**: This is the standard operational state of a Manufactured SP and defines the initial required access control settings of an SP based on the Templates incorporated into the SP, prior to personalization.

   The Manufactured state is Mandatory for the Admin SP.

Support for the Manufactured state is mandatory for the Key Per I/O SP.

5.2.2.2 State transitions for Manufactured SPs

The following sections describe the mandatory and optional state transitions for Key Per I/O SSC-compliant Manufactured SPs.

For the Admin SP, the only transition for which support is mandatory is "ANY STATE to ORIGINAL FACTORY STATE" (see section 5.2.2.2.2). As the only mandatory state for the Admin SP is Manufactured, the only mandatory transition is from Manufactured to Manufactured with the side effect of reverting the entire TPer to its Original Factory State. See section 5.1.2 for details.

Support for the "ANY STATE to ORIGINAL FACTORY STATE" transition (see section 5.2.2.2.2) is Mandatory for the Key Per I/O SP. Specifically, support for the transition from Manufactured to Manufactured-Inactive is Mandatory for the Key Per I/O SP. This transition is accomplished via the `Revert` method (see section 5.1.2).

Support for the "Manufactured-Inactive to Manufactured" transition (see section 5.2.2.2.1) is Mandatory for the Key Per I/O SP. This transition is accomplished via the `Activate` method (see section 5.1.1).

5.2.2.2.1 Manufactured-Inactive to Manufactured Transition

The triggers are:

- The `Activate` method (see section 5.1.1) is successfully invoked on the SP's object in the Admin SP's `SP` table.

The side effects are:

- The value in the LifeCycleState column of the SP's object in the Admin SP's `SP` table changes to Manufactured.
- The current SID PIN (C_PIN_SID) in the Admin SP is copied into the `PIN` column of Admin1's C_PIN credential (C_PIN_Admin1) in the activated SP. This allows for taking ownership of the SP with a known PIN credential.
- Any functionality enabled by the templates incorporated into the SP becomes active.

When the Key Per I/O SP transitions from the Manufactured-Inactive state to the Manufactured state (via invocation of the `Activate` method), the Storage Device SHALL cause all existing user data to no longer be accessible if the Key Per I/O capability applies to all Namespaces in the NVM (see section 5.12 for additional details). The mechanism for how the Storage Device makes data inaccessible is implementation specific.

A Key Per I/O SP is considered owned if it is not in the Manufactured-Inactive state.

5.2.2.2.2 ANY STATE to ORIGINAL FACTORY STATE Transition

The triggers are:

- `Revert` is successfully invoked on the SP.

The side effects are:

- The value in the LifeCycleState column of the SP's object in the Admin SP's `SP` table changes to the value of the SP's Original Factory State.
- The SP itself reverts to its Original Factory State, as described in section 5.1.2
- If the SP's Original Factory State was Manufactured-Inactive, any functionality enabled by the templates incorporated into the SP becomes inactive.

5.2.2.3 State behaviors for Manufactured SPs

5.2.2.3.1.1 Manufactured-Inactive

Any functionality enabled by the templates incorporated into the SP is inactive in the Manufactured-Inactive state. Sessions cannot be opened to SPs in this state.

When the Key Per I/O SP is in the Manufactured-Inactive state, the TCG management of the Storage Device's media encryption features (i.e., TCG management of the TPer's Key Per I/O SP functionality) SHALL be disabled.

5.2.2.3.1.2 Manufactured

Behavior of an SP in the Manufactured state is identical to the behavior of an SP in the Issued state, as described in [3].

When the Key Per I/O SP is in the Manufactured state, the TCG management of the Storage Device's media encryption features SHALL be enabled.

5.2.3 LifeCycle Type Modification

In order to accommodate the additional life cycle states defined in Key Per I/O, the life_cycle_state type for Key Per I/O SHALL be as defined in Table 61:

Table 61: LifeCycle Type Modification

UID	Name	Format	Size	Description
00 00 00 05 00 00 04 05	life_cycle_state	Enumeration_Type, 0, 15		Used to represent the current life cycle state.

Table 62 defines the valid enumeration values for the life_cycle_state type.

Table 62: LifeCycle Type Enumeration Values

Enumeration Value	Associated Value
0	Issued
1	Issued-disabled
2	Issued-frozen
3	Issued-disabled-frozen
4	Issued-failed
5-7	Reserved
8	Manufactured-inactive
9	Manufactured
10	Manufactured-disabled
11	Manufactured-frozen
12	Manufactured-disabled-frozen
13	Manufactured-failed
14-15	Reserved

5.3 Byte Table Access Granularity

Start of informative comment

While the general architecture defined in [3] allows data to be written into byte tables starting at any arbitrary byte boundary and with any arbitrary byte length, certain types of Storage Devices work more efficiently when data is written aligned to a larger block boundary. This section defines extensions to [3] that allow a Storage Device to report the restrictions that it enforces when the host invokes the `Set` method on byte tables.

End of informative comment

5.3.1 Table Table Modification

To allow a Storage Device to report its mandatory and recommended data alignment restrictions when accessing byte tables, the `Table` Table SHALL contain the additional columns shown in Table 63.

The mandatory and recommended data alignment restrictions do not apply to Object Tables.

Table 63: Table Table Additional Column

Column Number	Column Name	IsUnique	Column Type
0x0D	MandatoryWriteGranularity		uinteger_4
0x0E	RecommendedAccessGranularity		uinteger_4

5.3.1.1 MandatoryWriteGranularity

This column is used to indicate the granularity that the Storage Device enforces when the host invokes the `Set` method on byte tables.
The MandatoryWriteGranularity column SHALL NOT be modifiable by the host.

5.3.1.1.1 Byte Tables

For rows in the `Table` Table that pertain to byte tables, the MandatoryWriteGranularity column indicates the mandatory access granularity (in bytes) for the `Set` method for the table described in those rows of the `Table` Table. The MandatoryWriteGranularity column indicates the alignment requirement for both the access start offset (the Where parameter) and length (number of bytes in the Values parameter).

The value of the MandatoryWriteGranularity column SHALL be less than or equal to the value in the RecommendedAccessGranularity column in the same row of the `Table` Table.

The value of MandatoryWriteGranularity SHALL be less than or equal to 8192.

When the host invokes the `Set` method on a byte table, if ValidMandatoryGranularity (see Table 64) is False, then the method SHALL fail with status INVALID_PARAMETER.

If a TPer does not have a requirement on mandatory alignment for the byte table described in a row of the `Table` Table, then its MandatoryWriteGranularity column SHALL be set to one.

Table 64: ValidMandatoryGranularity Definition

For the `Set` method: ValidMandatoryGranularity is True if a) (x modulo MandatoryWriteGranularity) = 0 and b) (y modulo MandatoryWriteGranularity) = 0 where: x = the start offset of the `Set` method (i.e., the value of the Where parameter) y = the number of data bytes being set (i.e., the length of the Values parameter)

5.3.1.1.2 Object Tables

For rows in the `Table` Table that pertain to object tables, the value of the MandatoryWriteGranularity column SHALL be zero.

5.3.1.2 RecommendedAccessGranularity

This column is used to indicate the granularity that the Storage Device recommends when the host invokes the Set or `Get` method on byte tables.

The RecommendedAccessGranularity column SHALL NOT be modifiable by the host.

5.3.1.2.1 Byte Tables

For rows in the `Table` Table that pertain to byte tables, the RecommendedAccessGranularity column indicates the recommended access granularity (in bytes) for the `Set` and `Get` method for the table described in these rows of the `Table` Table. The RecommendedAccessGranularity column declares the alignment of data for the `Set` and `Get` method that allows for optimal `Set`/`Get` performance.

If a TPer does not have a recommended alignment for the byte table described in a row of the `Table` Table, then its RecommendedAccessGranularity column SHALL be set to one.

When the host invokes the `Set` method on a byte table, if ValidRecommendedGranularity (see Table 65) is False, then the performance of the TPer MAY be reduced when processing the method.

Table 65: ValidRecommendedGranularity Definition for Set

For the `Set` method: ValidRecommendedGranularity is True if a) (x modulo RecommendedAccessGranularity) = 0 and b) (y modulo RecommendedAccessGranularity) = 0 where: x = the start offset of the `Set` method (i.e., the value of the Where parameter) y = the number of data bytes being set (i.e., the length of the Values parameter)

When the host invokes the `Get` method on a byte table, if ValidRecommendedGranularity (see Table 66) is False, then the performance of the TPer MAY be reduced when processing the method.

Table 66: ValidRecommendedGranularity Definition for Get

For the `Get` method: ValidRecommendedGranularity is True if a) (x modulo RecommendedAccessGranularity) = 0 and b) (y modulo RecommendedAccessGranularity) = 0 where: x = the start offset of the `Get` method (i.e., the value of the startRow component of the Cellblock parameter) y = the number of data bytes being retrieved (i.e., the difference of the endRow and startRow components of the Cellblock parameter, plus one)

5.3.1.2.2 Object Tables

For rows in the `Table` Table that pertain to object tables, the value of the RecommendedAccessGranularity column SHALL be zero.

5.4 Mapping between Key Per I/O and OASIS KMIP

Start of informative comment

As described in early sections of this specification, Key Per I/O SSC depends upon the KMIP protocol for the formatting and transportation of keys from the host to a Storage Device.

This section provides users and implementers of Key Per I/O with KMIP with a limited set of the KMIP protocol elements (i.e., operations, objects, attributes, and error statuses) that are required to correctly implement Key Per I/O's key injection functionalities and the ways in which those protocol elements should be used to comply with this specification.

In addition, this section specifies how a Storage Device must interpret KMIP objects, attributes, and operations when they are invoked to implement various Key Per I/O functionality.

Readers are strongly encouraged to familiarize themselves with the KMIP specification (see [9]) and the KMIP Usage guide (see [10]) prior or during the reading of this section to understand Key Per I/O Storage Device's KMIP processing rules specified in this section.

End of informative comment

5.4.1 Minimum KMIP Specification Version

A Key Per I/O SSC compliant Storage Device SHALL support version 2.0 or greater of the KMIP specification (see [9]).

5.4.2 Required KMIP Capabilities

The set of KMIP capabilities required to implement Key Per I/O functionality in a manner compliant to this specification is indicated below. Support for KMIP capabilities that are not shown here for Key Per I/O usage is vendor implementation specific and outside the scope of this specification.

5.4.2.1 KMIP Communication Protocol

A Key Per I/O SSC compliant Storage Device SHALL implement support for the following KMIP protocol's elements:

a) KMIP Request and Response Message structure.

b) All required fields of the KMIP Request Message data structure, including required fields for the Request Header and Request Batch Item as specified by the KMIP specification (see [9] for details).

c) All required fields of the KMIP Response Message data structure, including all required fields for the Response Header and Response Batch Item as specified by the KMIP specification (see [9] for details).

   a. If a TPer does not support a real time clock, the Time Stamp value field SHALL always be set to 0.

d) All fields in KMIP Messages generated by a TPer SHALL appear in the order in which they are specified in the KMIP specification (see [9] for details).

e) Message Protocols:

   a. TTLV and appropriate padding where required as specified by the KMIP specification (see [9] for details).

5.4.2.2 KMIP Objects

A Key Per I/O SSC compliant Storage Device SHALL implement support for the following KMIP objects:

a) Symmetric Key object that includes:

   b. Key Block that includes:

      i. Key Format Type with values

         1. Raw

      ii. Key Value

      iii. Key Wrapping Data with values

         1. Wrapping Method with values

            a. Encrypt only

         2. Encryption Key Information with values

            a. Unique Identifier

b. Cryptographic Parameters with values
    i. Block Cipher Mode with values
        1. NIST Key Wrap if the Storage Device supports it in Level 0 (see section 3.1.1.3.16)
        2. GCM if the Storage Device supports it in Level 0 data (see section 3.1.1.3.17)
c. Padding Method with value set to OAEP if the Storage Device supports it in Level 0 data (see section 3.1.1.3.18)
d. Mask Generator Hashing Algorithm with value SHA-256 if the Storage Device supports the RSA-OAEP wrapping method (NOTE: The enumerated values of the Mask Generator Hashing Algorithm tag are the same as the enumerated values of the Hashing Algorithm in the KMIP specification).

3. IV/Counter/Nonce if the Storage Device reports support for key wrapping algorithm that requires it (see section 3.1.1.3.17)

5.4.2.3 KMIP Operations

A Key Per I/O SSC compliant Storage Device SHALL implement support for the following KMIP operations with their required fields as defined in [9] :

a) Discover versions;
b) Import; and
c) Query

Start of informative comment

A host uses the KMIP Query operation to interrogate a Key Per I/O Storage Device to determine KMIP protocol elements that the Storage Device implements. As defined in [9], a host's Query operation request is expected to specify a "Query Function" that maps to a set of KMIP protocol elements that the host is interested in. For example, if a host wishes to discover all KMIP Operations supported by a Key Per I/O compliant Storage Device, the host will specify a Query Function with a value of "Operations" in the Query request to the Storage Device. Then, the Storage Device will respond with a list of all KMIP Operations that it implements, including the Query operation itself (see section 5.4.4.3.2 for additional details).

End of informative comment

A Key Per I/O SSC compliant Storage Device SHALL implement support for Query with the following Query Function values:

a) Operations
b) Object Type

Support for other Query Function values is vendor implementation specific and is outside the scope of this specification.

5.4.2.4 KMIP Operations Data Structures

A Key Per I/O SSC compliant Storage Device SHALL implement support for the following KMIP operations data structure

a) Authentication Encryption Tag if the Storage Device supports a key wrapping algorithm that requires it (see section 3.1.1.3.17)

5.4.2.5 KMIP Object Attributes

A Key Per I/O SSC compliant Storage Device SHALL implement support for the following KMIP object attributes and their required fields as specified by KMIP specification (see [9] for details).

a) Cryptographic parameters attribute that includes:
   a. Key Role Type with values:
      i. KEK if a KEK Key is being injected; or
      ii. DEK if an MEK key is being injected
   b. Cryptographic Algorithm with value set to AES
   c. Cryptographic Length with values matching what's reported in the Level 0 Feature Descriptor data

b) Vendor attribute that includes the following:
   a. Vendor Identification with value always set to "TCG-SWG"
   b. Attribute Name
   c. Attribute Value

c) Link attribute with the following Link Type values
   a. Next Link
   b. Previous Link

5.4.3 Key Per I/O and OASIS KMIP Terminology Mapping

Table 67 below maps selected KMIP concepts relevant to Key Per I/O functionality onto terminology that is commonly used in the Key Per I/O SSC specification.

Where applicable, examples are given to help illustrate how a particular KMIP concept is used to implement Key Per I/O functionality.

Table 67: Key Per I/O and KMIP Terminology Mapping

OASIS KMIP TERM	KMIP VALUE FOR KEY PER I/O	KMIP TYPE	KEY PER I/O SSC TERM / KEY PER I/O SSC USAGE
Operation	Import	"Enumeration"	Command used to inject a key into a Storage Device.
	Query		Command used to discover KMIP protocol elements supported by the Storage Device and other default information for objects supported by the Storage Device
	Get		Command used to retrieve nonce objects from the Storage Device
Key Role Type	"KEK" or "DEK"	"Enumeration"	Attribute used to inform the Storage Device of the type of key being injected (see section 5.4.4.2.2.1 for details).
Vendor Identification Attribute	"TCG_SWG"	"Text String"	ID that alerts the Storage Device that the KMIP Request has TCG-SWG specific attributes processed according to TCG-SWG defined rules.

OASIS KMIP TERM	KMIP VALUE FOR KEY PER I/O	KMIP TYPE	KEY PER I/O SSC TERM / KEY PER I/O SSC USAGE
Attribute Name	“NamespaceID” or “Key Tag” or “UID”		“Namespace ID” and “Key Tag” values are used in an MEK injection request to indicate that the key is intended for a particular Key Tag slot within the Namespace specified by the “Namespace ID” value “UID” value is used for KEK injection to indicate the TCG UID in the `KeyEncryptionKey` Table the Storage Device should associate the injected key with.
Attribute Value	“1” or “$KEK_OBJ_UID”	“Integer”	Used together with Attribute Name to indicate the actual value of the Attribute Name value (see section 5.4.4.2.2.2.2 for details)
Link Type	Next Link	“Enumeration”	Used for MEK injections only to indicate which XTS-AES key is Key1 or Key2 (see section 5.4.4.2.2.3 for details)
	Previous Link		
Linked Object Identifier	“$KMIP_Key_UID”	“Text String”	Used together with “Link Type” in MEK injections For MEKs, if used with “Next Link” Link Type, this UID points to the KMIP KeyUID of the XTS-AES Key2. For MEKs, if used with “Previous Link” Link Type, this UID points to the KMIP KeyUID of the XTS-AES Key1.
Cryptographic Usage Mask	Encrypt Decrypt	“Enumeration”	Used for MEK injection only. Used to indicate that injected MEK should be used for both read host I/O and write host I/O.
	Unwrap		Used for KEK injection only. Used to indicate that the injected KEK should only be used for unwrapping other keys
Unique Identifier	$KMIP_Key_UID”	“Text String”	Globally unique identifier of a key object.
Key Wrapping Data	Wrapping Method	Structure	Used to indicate the method used for wrapping the key value
	Encryption Key Information	Structure	Used to indicate the KMIP KeyUID associated with the key that was used to encrypt the key object
	IV/Counter/Nonce	Byte String	Used to communicate the initialization vector or nonce if required by algorithm used to wrap the key value
Cryptographic Parameters (used	Block Cipher Mode if AES Key. Padding Method;	Structure	Used for MEK or KEK injections to indicate the wrapping method’s cryptographic details that the Storage Device uses to unwrap the injected key

OASIS KMIP TERM	KMIP VALUE FOR KEY PER I/O	KMIP TYPE	KEY PER I/O SSC TERM / KEY PER I/O SSC USAGE
in a Key Wrapping Data structure)	Mask Generator Hashing Algorithm		(see section 3.1.1.3.16 or 3.1.1.3.17 for supported values)
Cryptographic Parameters (used outside of Key Block)	Key Role Type; Cryptographic Algorithm; Cryptographic Length	Structure	Used to indicate to a TPer the cryptographic properties of the key being injected.
Authenticated Encryption Tag	$Tag_Value	Byte String	Used for MEK or KEK injections to specify the tag needed to authenticate wrapped data if the algorithm used to wrap the key value outputs it.
Failure Result Reason	Invalid Message; Unsupported Protocol Version; Server Limit Exceeded; Operation Denied; Invalid Attribute Value; etc..	Enumeration	KMIP term for error codes / error statuses.

5.4.4 Key Per I/O Storage Device Interactions with KMIP

Start of informative comment

The following rules specify how a Storage Device should interpret KMIP objects, attributes, and operations as they are invoked to implement Key Per I/O functionality.

End of informative comment

5.4.4.1 Interactions with KMIP Request Message and Response Message Data Structures

5.4.4.1.1 Request Message Header

Start of informative comment

Figure 12 shows a typical request message header for a Key Per I/O KMIP request.

For Key Per I/O, all requests generally use a similar header structure as the header does not contain any information that is specific to a particular Key Per I/O functionality. Differences in header composition may occur if the request contains multiple Batch Items and the host wants a TPer to process them in a specific order.

As specified in section 5.4.2.1, required elements of the KMIP Request Message structure, including the header, need to be declared in the request and, consistent with KMIP requirements, appear in the order they are specified in the KMIP specification (see [9] for details).

```
<KMIP>
  <!--  Inject plaintext KEK into TPer Through KMIP  -->
  <RequestMessage>
    <!--  Request Header  -->
    <RequestHeader>
      <!--  Protocol version required  -->
      <ProtocolVersion>
        <ProtocolVersionMajor type="Integer" value="2"/>
        <ProtocolVersionMinor type="Integer" value="1"/>
      </ProtocolVersion>
      <!--  BatchErrorContinuationOption shall not be present if BatchCount == 1  -->
      <!--  BatchOrderOption is optional if BatchCount == 1.  -->
      <!--  BatchOrderOption is True by default  -->
      <!--  BatchCount required  -->
      <BatchCount type="Integer" value="1"/>
    </RequestHeader>
```

Figure 12: Key Per I/O KMIP Request Header

End of informative comment

If a TPer receives a Key Per I/O KMIP Request Message where required elements of the Request Header are not declared or appear in an order not specified in the KMIP specification, the TPer SHALL fail the Batch Items in the request, up to the TPer's supported Protocol3MaxKmipBatchItems (see section 4.1.1.1.2), with Invalid Message Result Reason.

5.4.4.1.1.1 Protocol Version

As specified in section 5.4.1, a Key Per I/O compliant Storage Device supports a version 2.0 or greater of the KMIP specification. If the Storage Device receives a KMIP Request Message where the Protocol Version is less than protocol version major 2, the Storage Device SHALL fail the KMIP Request Message with Unsupported Protocol Version Result Reason. If the request contains multiple Batch Items, the TPer SHALL fail the Batch Items in the request, up to the TPer's supported Protocol3MaxKmipBatchItems (see section 4.1.1.1.2), with Unsupported Protocol Version Result Reason.

5.4.4.1.1.2 Batch Items

To minimize the number of host-to-Storage Device trips during certain Key Per I/O operations, a Key Per I/O compliant Storage Device SHALL support the capability to:

1. Accept multiple Batch Items within a single KMIP Request Message; and
2. Create multiple Batch items within a single KMIP Response Message

The Storage Device SHALL process KMIP Request Message whose Batch Count value, as indicated in the Request Message's Request Header, is less than or equal to the Protocol3MaxKmipBatchItems property that the Storage Device advertises (see section 4.1.1.1). Otherwise, the Storage Device SHALL fail the Batch Items in the request, up to the TPer's supported Protocol3MaxKmipBatchItems (see section 4.1.1.1.2), with Server Limit Exceeded KMIP Result Reason.

A Key Per I/O compliant Storage Device SHALL support all required fields of the Batch Item data structure in the order specified by the KMIP specification.

If a TPer receives a Key Per I/O KMIP Request Message where the required elements of the Request Batch Item are not declared or appear in an order not specified in the KMIP specification, the TPer SHALL fail the Batch Item operation in the request with Invalid Message Result Reason.

5.4.4.2 Key Injections Interactions

Key Per I/O's KEK or MEK injections are accomplished using KMIP's Import operation. As specified in section 5.4.2.3, a Key Per I/O compliant Storage Device is capable of processing Import operation as defined in [9].

5.4.4.2.1 KMIP Key Injection Interface Locking

Start of informative comment

The ability to provision keys in a Storage Device using the KMIP key injection interface is a cryptographic service that is offered and controlled by a TPer's Admin authorities.

Via policy definition (see section 4.3.5.1), the Admin authorities can control when keys can be injected into a Storage Device and can specify reset events that could block key injection interfaces until Admin authorities are ready to authorize keys injections again. This mechanism can help prevent a malicious host entity from potentially provisioning its own keys which could put the availability of user data on a Storage Device at risk.

End of informative comment

A Key Per I/O compliant Storage Device SHALL support the ability to disable and enable KMIP's key injection interfaces (i.e., processing of Import Operation) as specified by the `KPIOPolicies` Table (see section 4.3.5.1 for details).

If a TPer receives a KMIP Batch Item request where:

a) The value specified for operation field is Import;
b) the column value of the `KeyInjectionInterfaceLockEnabled` column in the `KPIOPolicies` Table (see section 4.3.5.1.7) is set to True;
c) the column value of the `KeyInjectionInterfaceLocked` column in the `KPIOPolicies` Table (see section 4.3.5.1.8) is set to True; and
d) the command would otherwise succeed,

then the TPer SHALL fail the Batch Item operation with Operation Denied Result Reason.

5.4.4.2.2 Interactions with KMIP Symmetric Key's Object Attributes

Start of informative comment

As described in section 5.4.2.5, Key Per I/O leverages KMIP attributes to communicate properties associated with symmetric key objects being injected into a TPer. Examples include cryptographic properties such as key length, key wrapping information, etc.

In addition, Key Per I/O leverages attributes such as Vendor Attribute to communicate TCG SWG-specific Key Per I/O concepts that are not natively supported by KMIP. Examples include concepts such as TCG Objects' UIDs, Key Tags, and Namespace IDs.

This section defines which attributes and attributes' values are required for a particular type of key injection request (KEK injection vs. MEK injection) and applicable restrictions.

Additionally, this section defines rules that govern how a Key Per I/O compliant Storage Device should interpret these KMIP attributes and their values when they are used to specify Key Per I/O data.

End of informative comment

5.4.4.2.2.1 Cryptographic Parameters Attribute

Start of informative comment

To facilitate optimized parsing of KMIP key injection requests on a Storage Device, Key Per I/O leverages KMIP's Cryptographic Parameters attribute when used in the Request payload's attributes section outside of the Key Block structure, to communicate to a TPer the basic cryptographic properties of the keys being injected, such as the

role/type of the key and its cryptographic length and algorithm.

The ability to communicate the type/role of the key early in the request enables a Storage Device to efficiently parse and process the rest of key injection request message without solely relying on the properties of the keys to comprehend that the key being injected is a KEK or an MEK.

The Cryptographic Algorithm and Cryptographic Length of the key being injected do not necessarily need to be specified within the Cryptographic Parameters attribute in the Request Payload's Attributes section. They can also be specified inside the Key Block structure outside the Key Value (see [10] for details).

In addition, when the injected key is wrapped, the Cryptographic Parameters attribute is repeatedly used inside the Key Block's Key Wrapping Data's Encryption Key Information structure to communicate cryptographic properties of the key used for wrapping the injected key. When used this way, the Cryptographic Parameters' fields specify the Cryptographic Algorithm used to wrap the key, the Block Cipher Mode if the wrapping method was AES-based, or the Padding Method if RSA was used for wrapping. Additional information such as the Mask Generator Hashing Algorithm to use with RSA-OAEP or the IV/Counter/Nonce to use with AES-GCM need to be specified in the Cryptographic Parameters, as well depending on the wrapping algorithm used.

End of informative comment

A Key Per I/O compliant Storage Device SHALL support the values of Cryptographic Parameters' Key Role Type attribute defined in Table 68 as follows:

Table 68: Object Group Values Definition for Symmetric Key Object

Key Role Type	Encoding	TPer's Expected Behavior
KEK	Enumeration	A TPer SHALL process an object that specifies this value as a Key Encryption Key object intended for the `KeyEncryptionKey` Table.
DEK	Enumeration	A TPer SHALL process a symmetric key object that specifies this value as one component of the XTS-AES MEK (i.e., as Key1 or Key2) intended for the Storage Device's key cache.

All symmetric key injection requests are expected to specify the Cryptographic Parameters' Key Role Type attribute and its appropriate values as defined by Table 68 in the Import operation's Request Payload's Attributes section outside of the Key Block structure.

If a Storage Device receives a KMIP Request Message's Batch Item to import a symmetric key where the Key Role Type and its value are not specified or are not specified in the Attributes section outside the Key Block structure, the Storage Device SHALL fail the Batch Item's operation with Invalid Message Result Reason.

5.4.4.2.2.2 Vendor Attribute

Start of informative comment

Key Per I/O SSC leverages the KMIP's Vendor Attribute to define TCG-SWG specific attributes and TCG-SWG specific processing rules tied to those attributes.

Specifically, Key Per I/O SSC allows for Vendor Attribute's Attribute Name and Attribute Value fields to be used in conjunction with the Vendor Attribute's Vendor Identification to indicate TCG SWG-specific Key Per I/O information such as the Namespace, the Namespace's key tag, or a TCG UID for which the injected keys are intended, so the Storage Device can process them as intended by TCG SWG.

Figure 13 shows an example of how the Vendor Attribute is used in the Request Payload's Attributes section of a KMIP Import operation to inject an MEK key object associated with a particular Namespace ID and a particular Key Tag within that Namespace.

```
  ..................
  <!--  Note: Specify the NamespaceID associated with row in KeyTagAllocationTable table  -->
▼<Attribute>
    <VendorIdentification type="TextString" value="$TCG-SWG"/>
    <AttributeName type="TextString" value="NamespaceID"/>
    <AttributeValue type="Integer" value="1"/>
  </Attribute>
  <!--  Note: Specify the Key Tag Slot within the specified NamespaceID -->
▼<Attribute>
    <VendorIdentification type="TextString" value="$TCG-SWG"/>
    <AttributeName type="TextString" value="KeyTag"/>
    <AttributeValue type="Integer" value="1"/>
  </Attribute>
  ..................
```

Figure 13: KMIP's Vendor Identification Usage for Key Per I/O

End of informative comment

5.4.4.2.2.1 Vendor Identification

As Figure 13 shows, KMIP Vendor Identification Attribute is expected be specified for all Key Per I/O KMIP Request Messages injecting keys. In addition, this attribute is expected to always be used with its value field set to "TCG-SWG".

If a Storage Device receives a KMIP Request Message's Batch Item for injecting a key that does not include this attribute in Attributes section, or a request that includes it but with its value field set to something other than "TCG-SWG", the Storage Device SHALL fail the KMIP Request Message's Batch Item operation with KMIP's Invalid Message Result Reason.

5.4.4.2.2.2 AttributeName and AttributeValue

Start of informative comment

KMIP's Vendor Attribute's AttributeName and AttributeValue are used to communicate to a TPer the values of the following TCG-SWG specific Key Per I/O concepts: TCG KEK Object UID, Namespace ID, Key Tag.

End of informative comment

A Key Per I/O SSC compliant Storage Device SHALL implement the following behavior when KMIP's Vendor Attribute's Attribute Name and Attribute Value are used as described in Table 69:

Table 69: KMIP's Vendor Attribute's Attribute Name and Attribute Value Usage with Key Per I/O

Attribute Name	Attribute Value's KMIP Type Encoding	TPer's Expected Behavior	Key Role Type In Scope for Attribute Usage
UID	Byte String	Specifies the TCG KEK Object's UID that a TPer SHALL associate with the injected key in the `KeyEncryptionKey` Table.	A KEK is always expected to specify this entry.
NamespaceID	Integer	Specifies the Namespace ID that a TPer SHALL associate with the injected key.	

Attribute Name	Attribute Value's KMIP Type Encoding	TPer's Expected Behavior	Key Role Type In Scope for Attribute Usage
UID	Byte String	Specifies the TCG KEK Object's UID that a TPer SHALL associate with the injected key in the KeyEncryptionKey Table.	A KEK is always expected to specify this entry.
KeyTag	Integer	Specifies the Key Tag value in specified Namespace ID that a TPer SHALL associate with the injected key.	A DEK is always expected to specify these attributes.

Start of informative comment

The following rule defines how a TPer should associate a KEK with a TCG KEK UID value specified by the Vendor attribute.

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

1. the KEK Key Role Type is specified;
2. the AttributeName is specified;
3. the AttributeName value specifies "UID";
4. the AttributeValue is specified;
5. the value specified by AttributeValue matches a UID associated with a row in the KeyEncryptionKey Table; and
6. the operation would otherwise complete successfully,

then, the TPer SHALL associate the injected key object with a KeyEncryptionKey object associated with the row specified by that UID value and SHALL complete the Batch Item's operation successfully.

Start of informative comment

The following rule prevents a TPer from associating a KEK key object with a UID in the KeyEncryptionKey Table if the UID specified by the AttributeValue does not exist.

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

1. the KEK Key Role Type is specified;
2. the AttributeName is specified;
3. the AttributeName value specifies "UID";
4. the AttributeValue is specified;
5. the value specified by AttributeValue does not match a TCG UID associated with a row in the KeyEncryptionKey Table; and
6. the operation would otherwise complete successfully,

then, the TPer SHALL fail the Batch Item operation with Invalid Attribute Value Result Reason.

Start of informative comment

The following rule defines how a TPer should associate an MEK with a Namespace ID value specified by the Vendor attribute.

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

1. the DEK Key Role Type is specified;
2. the AttributeName is specified;
3. the AttributeName value specifies "NamespaceID",
4. the AttributeValue is specified;
5. the value specified by the Namespace ID's specified AttributeValue matches a Namespace ID associated with a `KeyTagAllocation` Table row with the `Managed` column set to TRUE (see section 4.3.5.2.5 for details); and
6. the operation would otherwise complete successfully,

then, the TPer SHALL associate the injected key object with the Namespace specified by the value of the AttributeValue and SHALL complete the Batch Item's operation successfully.

Start of informative comment

The following rule prevents a TPer from associating an MEK with a Namespace that is not managed by Key Per I/O.

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

1. the DEK Key Role Type is specified;
2. the AttributeName is specified;
3. the AttributeName value specifies "NamespaceID";
4. the AttributeValue is specified;
5. the value specified by the Namespace ID's specified AttributeValue matches a Namespace ID associated with a `KeyTagAllocation` Table row with the `Managed` column set to FALSE (see section 4.3.5.2.5 for details); and
6. the operation would otherwise complete successfully,

then, the TPer SHALL fail the Batch Item operation with Permission Denied Result Reason.

Start of informative comment

The following rule prevents a TPer from associating an MEK with an invalid or inexistent Namespace ID.

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

1. the DEK Key Role Type is specified;
2. the AttributeName is specified;
3. the AttributeName value specifies "NamespaceID";

4. the AttributeValue is specified;
5. the value specified by the Namespace ID's specified AttributeValue does not match any row in the KeyTagAllocation Table;
6. the operation would otherwise complete successfully,

then, the TPer SHALL fail the Batch Item operation with Invalid Attribute Value Result Reason

Start of informative comment

The following rule defines how a TPer should associate an MEK with a specified key tag value that is within the host configured limit of the number of key tags allocated for the specified Namespace.

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

1. the DEK Key Role Type is specified;
2. the AttributeName is specified;
3. the AttributeName value specifies "NamespaceID";
4. the value specified by the Namespace ID's specified AttributeValue matches a Namespace associated with a `KeyTagAllocation` Table row with the `Managed` column set to TRUE (see section 4.3.5.2.5 for details);
5. another AttributeName is specified;
6. the AttributeName value specifies "KeyTag";
7. the AttributeValue is specified;
8. the value specified by the Key Tag's specified AttributeValue is less than the value of the `NumberOfKeyTags` column for the `KeyTagAllocation` Table row associated with the specified NamespaceID' s AttributeValue; and
9. the operation would otherwise complete successfully,

then, the TPer SHALL associate the injected key object with the value of the specified Key Tag's AttributeValue within the Namespace specified by the value of the NamespaceID' s AttributeValue and SHALL complete the Batch Item's operation successfully.

Start of informative comment

The following rule prevents a TPer from associating an MEK with a specified key tag value that is equal or greater than the number of key tags allocated for the specified Namespace. Because the Key Tag field is a 0-based index field, specifying a value that is equal to the number of key tags allocated for a Namespace is specifying a Key Tag value that is past the supported indexes.

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

1. the DEK Key Role Type is specified;
2. the AttributeName is specified;
3. the AttributeName value specifies "NamespaceID";
4. the value specified by the Namespace ID's specified AttributeValue matches a Namespace associated with a `KeyTagAllocation` Table row with the `Managed` column set to TRUE (see section 4.3.5.2.5 for details);

5. another AttributeName is specified;
6. the AttributeName value specifies “KeyTag”;
7. the AttributeValue is specified;
8. the value specified by the Key Tag’s specified AttributeValue is equal to or greater than the value of the NumberOfKeyTags column for the KeyTagAllocation Table row associated with the specified NamespaceID’ s AttributeValue; and
9. the operation would otherwise complete successfully,

then, the TPer SHALL fail the Batch Item operation with Invalid Attribute Value Result Reason.

Start of informative comment

The following rule prevents a TPer from associating an MEK with a specified key tag value if the number of key tags allocated for the specified Namespace is 0.

End of informative comment

If a TPer receives a KMIP Batch Item’s Request Payload to import a symmetric key where:

1. the DEK Key Role Type is specified;
2. the AttributeName is specified;
3. the AttributeName value specifies “NamespaceID”;
4. the value specified by the Namespace ID’s specified AttributeValue matches a Namespace associated with a KeyTagAllocation Table row with the Managed column set to TRUE (see section 4.3.5.2.5 for details);
5. another AttributeName is specified;
6. the AttributeName value specifies “KeyTag”;
7. the AttributeValue is specified;
8. the value of the AttributeValue is specified;
9. the value of the NumberOfKeyTags column for the KeyTagAllocation Table row associated with the specified NamespaceID’ s AttributeValue is 0; and
10. the operation would otherwise complete successfully,

then, the TPer SHALL fail the Batch Item operation with Permission Denied Result Reason.

5.4.4.2.2.3 Vendor Attribute Usage for Replay Protection

Start of informative comment

Key Per I/O allows for KMIP’s Vendor attribute to be used to declare a Nonce that must be encrypted together with the key to allow detection of potential replay attacks involving that key.

As stated in [10] and depicted by Figure 14, a host is responsible for creating the custom Vendor attribute, assigning it to the key it wants retrieved, and populating the attribute with the most recent Nonce value retrieved from a TPer (see section 3.2.6 for details on retrieving a Nonce from a TPer).

```
<!--  Note: Create Custom Attribute using SetAttribute operation for KEK1  -->
<!--  Note Cont: prior to requesting the KEK1 wrapped with the attribute   -->
<BatchItem>
  <Operation type="Enumeration" value="SetAttribute"/>
  <UniqueBatchItemID type="ByteString" value="01"/>
  <RequestPayload>
    <UniqueIdentifier type="TextString" value="$UNIQUE_IDENTIFIER_KEK1"/>
    <NewAttribute>
      <Attribute>
        <VendorIdentification type="TextString" value="TCG-SWG"/>
        <!--  Note: Name the attribute so it can be referenced later -->
        <AttributeName type="TextString" value="KPIO-NONCE"/>
        <!--  Note: Specify Actual NONCE value to be encrypted with the key  -->
        <AttributeValue type="ByteString" value="$N_BYTES_NONCE_VALUE"/>
      </Attribute>
    </NewAttribute>
  </RequestPayload>
</BatchItem>
```

Figure 14: Replay Protection Example with Vendor Attribute – Adding Nonce to a Key

```
<!--  Note: Use Get Operation to retrieve KEK1  -->
<BatchItem>
  <Operation type="Enumeration" value="Get"/>
  <UniqueBatchItemID type="ByteString" value="02"/>
  <RequestPayload>
    <UniqueIdentifier type="TextString" value="$UNIQUE_IDENTIFIER_KEK1"/>
    <KeyWrappingSpecification>
      <WrappingMethod type="Enumeration" value="Encrypt"/>
      <!--  Note: Wrap the key using RSA-OAEP  -->
      <EncryptionKeyInformation>
        <UniqueIdentifier type="TextString" value="$KEY_PER_IO_CERT_OBJ_NAME"/>
        <CryptographicParameters>
          <PaddingMethod type="Enumeration" value="OAEP"/>
          <MaskGeneratorHashingAlgorithm type="Enumeration" value="SHA256"/>
        </CryptographicParameters>
      </EncryptionKeyInformation>
      <!--  Note: Reference attrib. whose value needs to be wrapped with KEK1 -->
      <Attribute>
        <VendorIdentification type="TextString" value="TCG-SWG"/>
        <AttributeName type="TextString" value="KPIO-NONCE"/>
      </Attribute>
      <!--  Note: NoEnconding means TTLV default encoding  -->
      <EncodingOption type="Enumeration" value="NoEncoding"/>
    </KeyWrappingSpecification>
  </RequestPayload>
</BatchItem>
```

Figure 15: Replay Protection Example with Vendor Attribute - Retrieving Key Wrapped with a Nonce

As described in [10] , the host is expected to associate the Nonce with the key before it requests the KMS to return the key wrapped together with that Nonce value.

Figure 15 depicts a Key that is retrieved wrapped with a Nonce. As Figure 15 shows, the host is expected to reference the AttributeName associated with the Nonce in Key Wrapping Specification's Attribute section as part of the Get operation to retrieve the key.

The host and the KMS will need to agree on some "processing rules" involving the usage of Vendor attribute so that the KMS will be able to know to retrieve the value associated with the named attribute, encrypt it together with the key material, and return the ciphered key value (and its authentication tag if an algorithm which generates authentication tag is used for encryption).

The resulting ciphered key value is TTLV-encoded structure parsed by a Storage Device as follows: Key Material Tag || Byte String || Length || Ciphered Key Material Value.

The first component of the Ciphered Key Material Value (i.e., from its first bit that appears in the sequence up to the specified Cryptographic Length of the key) represents the actual key data while the remaining bytes represent the Nonce value.

Replay protection is ensured by a Storage Device verifying that the Nonce value matches what it expects after validating the authentication tag. If the values match, the key will be accepted. If they don't match, an appropriate error will be returned (See section 4.3.5.1.4 for the appropriate error status).

End of informative comment

5.4.4.2.2.3 Link Attribute

Start of informative comment

Key Per I/O allows for KMIP's Link attribute to be used to specify a linkage between two keys that indicates the relationship between them.

For example, if the host is injecting an XTS-AES key, it will need to state which key is the XTS-AES Key1 and which key is the XTS-AES Key2. In such example, the host will create a Link attribute for those two keys with the appropriate Link Type to indicate which one is Key1 or Key2.

End of informative comment

A Key Per I/O SSC compliant Storage Device SHALL support the Linked Object Identifier values for the Link attribute as shown in Table 70.

Table 70: Supported LinkedObjectIdentifier Values for Symmetric Key Object

Supported KMIP's Linked Object Identifier Value	TPer's Expected Behavior	Key Role Type in Scope for Attribute Usage
Next Link	A TPer SHALL use the DEK key specifying this value as the XTS-AES Key1	A DEK is expected to always specify this attribute.
Previous Link	A TPer SHALL use the DEK key specifying this value as the XTS-AES Key2	

5.4.4.2.3 Key Encryption Keys Injection

Start of informative comment

Figure 16 shows an example of a typical plaintext KEK injection request using KMIP's Batch Item data structure. As the example shows, all the Batch Item's required fields need to be included and appear in the order in which they are defined in the KMIP specification, starting with the operation, a unique Batch Item ID if needed, and the Request Payload. The latter features the KMIP KeyUID associated with the KEK object being injected, its Object Type, and the Attributes associated with the KEK object.


```
<RequestHeader>
  <!--  Protocol version required  -->
  <ProtocolVersion>
    <ProtocolVersionMajor type="Integer" value="2"/>
    <ProtocolVersionMinor type="Integer" value="1"/>
  </ProtocolVersion>
  <!--  BatchErrorContinuationOption shall not be present if BatchCount == 1  -->
  <!--  BatchOrderOption is optional if BatchCount == 1.  -->
  <!--  BatchOrderOption is True by default  -->
  <!--  BatchCount required  -->
  <BatchCount type="Integer" value="1"/>
 </RequestHeader>
 <!--  Batch Item 1: Inject Plaintext KEK  -->
<BatchItem>
  <Operation type="Enumeration" value="Import"/>
  <!--  UniqueBatchItemID required if BatchCount > 1  -->
  <UniqueBatchItemID type="ByteString" value="01"/>
  <RequestPayload>
    <!--  Globally unique KMIP Key UID associated with KEK  -->
    <UniqueIdentifier type="TextString" value="$UNIQUE_IDENTIFIER_KEK_1"/>
    <ObjectType type="Enumeration" value="SymmetricKey"/>
    <!--  Note: Specifies KEK and its attributes  -->
   <Attributes>
      <!--  256 KEK Key Crypto Params  -->
    <CryptographicParameters>
       <KeyRoleType type="Enumeration" value="KEK"/>
       <CryptographicAlgorithm type="Enumeration" value="AES"/>
       <CryptographicLength type="Integer" value="256"/>
     </CryptographicParameters>
     <!--  Specify TCG KEK UID  -->
     <!--  In KeyEncryptionKey table to be associated with KEK  -->
    <Attribute>
       <VendorIdentification type="TextString" value="$TCG-SWG"/>
       <AttributeName type="TextString" value="UID"/>
       <AttributeValue type="ByteString" value="$KEK_OBJECT_UID_1"/>
     </Attribute>
    </Attributes>
    <!--  Note: Specify KEK key material  -->
   <SymmetricKey>
     <KeyBlock>
       <!--  Note: Refer to KMIP User Guide 3.26 for "Raw" Key Format Type  -->
       <KeyFormatType type="Enumeration" value="Raw"/>
       <!--  Note: KeyValue is a Structure since key is plaintext  -->
      <KeyValue>
        <KeyMaterial type="ByteString" value="$KEY_MATERIAL_KEK_1"/>
       </KeyValue>
       <!--  <CryptographicAlgorithm type="Enumeration" value="AES"/>  -->
       <!--  <CryptographicLength type="Integer" value="256"/>  -->
     </KeyBlock>
    </SymmetricKey>
  </RequestPayload>
</BatchItem>
```

Figure 16: Specifying KEK, Its Attributes, and Plaintext Key Data with KMIP's Batch Item

Though KMIP does not impose any restrictions on the order in which attributes declared outside of the Key Block need to appear, it is strongly recommended that the host lists attributes in the order they are shown in Figure 17 in order to take advantage of potential optimizations for KMIP parsing that may exist in a Storage Device.

Once Attributes are declared, the key data follows as shown in Figure 16. As the figure shows, some attributes can also be included inside the Key Block. For example, as discussed in section 5.4.4.2.2.1, Cryptographic Algorithm and Cryptographic Length attributes are shown included (but commented out) inside the Key Block instead of the inside of the Cryptographic Parameters attribute in the Request Payload's Attributes section.

When the key being injected is wrapped, the Key Block structure needs to include the Key Wrapping Data that informs the Storage Device how the injected key was wrapped and the key to use to unwrap it. Figure 17 below shows an example.

```
    <!--  Note: Specify KEK key material  -->
  ▼<SymmetricKey>
   ▼<KeyBlock>
      <KeyFormatType type="Enumeration" value="Raw"/>
      <!--  Note: KeyValue is a ByteString since key is wrapped  -->
      <KeyValue type="ByteString" value="$WrappedKEK1"/>
      <!--  <CryptographicAlgorithm type="Enumeration" value="AES"/>  -->
      <!--  <CryptographicLength type="Integer" value="256"/>  -->
     ▼<KeyWrappingData>
        <!--  Note: Method used to wrap Key Value  -->
        <WrappingMethod type="Enumeration" value="Encrypt"/>
       ▼<EncryptionKeyInformation>
          <!--  Note: KMIP Key UID of KEK used to wrap injected key  -->
          <UniqueIdentifier type="TextString" value="$UNIQUE_IDENTIFIER_KEK_0"/>
         ▼<CryptographicParameters>
            <!--  Note: Crypto Algorithm used to wrap key  -->
            <CryptographicAlgorithm type="Enumeration" value="AES"/>
            <BlockCipherMode type="Enumeration" value="NISTKeyWrap"/>
          </CryptographicParameters>
        </EncryptionKeyInformation>
      </KeyWrappingData>
    </KeyBlock>
  </SymmetricKey>
 </RequestPayload>
</BatchItem>
```

Figure 17: Specifying Key Data for Wrapped KEK

Similar to the attributes declared outside the Key Block structure, attributes included within the Key Block structure need to be declared correctly and appear in the order in which they are defined in the KMIP specification, in order to guarantee that a TPer processes the injected key correctly.

End of informative comment

A TPer SHALL validate that a Batch Item request from the host to inject a KEK is structured as follows:

- The stated KMIP operation SHALL be an Import operation.
- If multiple KEKs are being injected, a unique identifier of each KMIP Batch Item SHALL be declared.
- The KMIP's Message Request Payload SHALL include:
  - The KMIP Unique Identifier of the key being injected with length less than or equal to the maximum value supported by the TPer (see section 3.1.1.3.14 for details).

- The SymmetricKey Object type
- An Attributes section that SHALL include:
  - Cryptographic Parameters attribute stating:
    - The Key Role Type with value set to KEK
    - The Cryptographic Algorithm with value set to AES
    - The Cryptographic Length with a value field set to what's supported by the Storage Device for AES-based key wrapping algorithms reported by Level 0 data
      - Optionally, Cryptographic Algorithm and Cryptographic Length attributes MAY be specified inside the Key Block structure outside of the key value.
  - The Vendor Attribute that SHALL include:
    - The Vendor Identification with the value field set to TCG-SWG
    - The Attribute Name and Attribute Value stating the UID of the KeyEncryptionKey object in the `KeyEncryptionKey` Table with which the TPer SHALL associate the injected key.
- A SymmetricKey structure that SHALL include:
  - A Key Block structure that SHALL include:
    - The Key Value (a structure if the injected key is plaintext, otherwise a Byte String):
    - The Key Wrapping Data structure if the injected key is encrypted, that SHALL include:
      - The Wrapping method with its value field set to Encrypt
      - An Encryption Key Information structure that SHALL include:
        - The Unique Identifier of the key previously provisioned into the TPer that was used to wrap the injected key.
          - If the Storage Device's Key Per I/O Public Key Certificate was used to encrypt the injected key, the value of the Unique Identifier field SHALL be set to the name of the Certificate1 object of the `Certificates` Table.
        - The Cryptographic Parameters that SHALL include:
          - A Cryptographic Algorithm set to AES or RSA depending on the wrapping method used to protect the injected key.
          - The Block Cipher Mode if the wrapping algorithm is AES based, with its value field set to NIST Key Wrap or GCM depending on what the Storage Device supports.
          - The Padding Method with its value field set to OAEP if RSA-OAEP was used to wrap the injected key.
          - The Mask Generator Hashing Algorithm with its value field set to SHA-256 if RSA-OAEP was used to wrap the injected key.

- An IV/Counter/Nonce if AES-GCM was used to wrap the key

- An Authenticated Encryption Tag if the injected key is wrapped and AES-GCM was used to wrap the key.

Start of informative comment

The following rule defines how a TPer extracts the Nonce from the ciphered key material if the Replay Protection feature has been enabled by the host on the TPer (see section 4.3.5.1.4).

End of informative comment

If Replay Protection has been enabled on a TPer, the TPer SHALL process the ciphered TTLV-encoded KEK key value as follows: Key Material Tag || Byte String || Length || Ciphered Key Material Value, where the first component of the Ciphered Key Material Value (i.e., from its first bit that appears in the sequence up to the declared Cryptographic Length of the key) SHALL represent the actual key data while the remaining bytes represent the Nonce value.

Start of informative comment

The following rule prevents a TPer from processing a KMIP operation to inject a KEK unless the required KMIP operation, object, or attributes and their values are as specified in section 5.4.4.2.3.

End of informative comment

If a Storage Device receives a KMIP Request Message's Batch Item to import a symmetric key object where a KEK Key Role Type is specified and all the required fields as specified in section 5.4.4.2.3 are not declared, the Storage Device SHALL fail the Batch Item operation with Invalid Message Result Reason.

Start of informative comment

The following rule prevents a TPer from processing a KMIP operation to inject a KEK if the Request Payload specifies attributes that are not applicable to a KEK Key Role Type. For example, specifying a Namespace ID or a Key Tag in addition to specifying a UID in the `KeyEncryptionKey` Table will result in an error.

End of informative comment

If a Storage Device receives a KMIP Request Message's Batch Item to import a symmetric key object where a KEK Key Role Type is declared and the Batch Item's payload includes attributes entries that are only applicable to a different Key Role Type value, the Storage Device SHALL fail the Batch Item operation with Invalid Message Result Reason

Start of informative comment

The following rule prevents a TPer from associating an injected KEK with a row in `KeyEncryptionKey` Table unless the injected KEK is wrapped with a key included in the allowed list of KEKs for the specified KEK UID.

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

1. the KEK Key Role Type is specified;
2. the AttributeName is specified;
3. the AttributeName value specifies "UID";
4. the AttributeValue is specified;

5. the specified value of the Unique Identifier field within the Encryption Key Information is associated with a key associated with row in the KeyEncryptionKey Table whose UID is not included in the list of UIDs of the AllowedKeyEncryptionKeys column for the row specified by the value of the AttributeValue; and
6. the operation would otherwise complete successfully,

then, the TPer SHALL fail the Batch Item operation with Permission Denied Result Reason

Start of informative comment

The following rule prevents a TPer from associating an injected KEK with a row in KeyEncryptionKey Table if the injected KEK is wrapped with a key that is in an Access Locked state.

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

1. the KEK Key Role Type is specified;
2. the specified value of the Unique Identifier field within the Encryption Key Information is associated with a key associated with row in the KeyEncryptionKey Table whose UID is included in the list of UIDs of the AllowedKeyEncryptionKeys column for the row specified by the specified TCG KEK Object UID;
3. the specified value of the Unique Identifier field within the Encryption Key Information is associated with a key associated with a row in KeyEncryptionKey Table whose AccessLocked and AccessLockEnabled columns are both set to True; and
4. the operation would otherwise complete successfully,

then, the TPer SHALL fail the Batch Item operation with Permission Denied Result Reason

Start of informative comment

The following rule prevents a TPer from associating an injected KEK with a row in KeyEncryptionKey object if the injected KEK UID maps to a row that is associated with a KeyEncryptionKey object that is in an Access Locked state.

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

1. the KEK Key Role Type is specified;
2. the AttributeName is specified;
3. the AttributeName value specifies "UID";
4. the AttributeValue is specified;
5. the value specified by AttributeValue matches a TCG UID associated with a row in the KeyEncryptionKey Table whose AccessLocked column and AccessLockEnabled column are both set to TRUE; and
6. the operation would otherwise complete successfully,

then, the TPer SHALL fail the Batch Item operation with Permission Denied Result Reason

Start of informative comment

The following rule prevents a TPer from associating an injected KEK with a row in KeyEncryptionKey Table unless the injected KEK is wrapped with a key whose key value equals key value stored by the TPer.

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

1. the KEK Key Role Type is specified;
2. the specified value of the Unique Identifier field within the Encryption Key Information is associated with a key associated with a row in `KeyEncryptionKey` Table whose AccessLocked or AccessLockEnabled column is set to False;
3. the specified value of the Unique Identifier field within the Encryption Key Information is associated with a key associated with row in the `KeyEncryptionKey` Table whose UID is included in the list of UIDs of the AllowedKeyEncryptionKeys column for the row specified by the specified TCG KEK Object UID;
4. the operation would otherwise complete successfully; and
5. unwrapping fails,

then, the TPer SHALL fail the Batch Item operation with Cryptographic Failure Result Reason

Start of informative comment

The following rule prevents a TPer from associating an injected KEK with a row in `KeyEncryptionKey` Table unless the injected KEK is wrapped with a key that has already been injected into the TPer.

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

1. the KEK Key Role Type is specified;
2. the specified value of the Unique Identifier field within the Encryption Key Information is not associated with a key associated with a row in `KeyEncryptionKey` Table; and
3. the operation would otherwise complete successfully,

then, the TPer SHALL fail the Batch Item operation with Invalid Attribute Result Reason.

5.4.4.2.4 Media Encryption Keys Injection

Start of informative comment

MEK Key1 and Key 2 injection requests use the standard KMIP elements as used by KEK injection. The difference between a KEK and MEK injection request is in the attributes included in the Attributes section.

For example, Figure 18 below illustrates how a host specifies the attributes associated with the XTS-AES Key 1.

As Figure 18 illustrates, MEK key injection request needs to declare the attributes such as Namespace and Key Tag as shown with appropriate values so the TPer can map the injected MEKs in the correct locations within its key cache.

Though not required by KMIP, it is recommended that the host lists the attributes in the order they are shown here to optimize parsing of MEK injections in the Storage Devices.


```
<!--  Note: Specifies MEK and its attributes  -->
▼<Attributes>
 ▼<CryptographicParameters>
    <!--  Note: 256 MEK Key  -->
    <KeyRoleType type="Enumeration" value="DEK"/>
    <CryptographicAlgorithm type="Enumeration" value="AES"/>
    <CryptographicLength type="Integer" value="256"/>
  </CryptographicParameters>
  <!--  Note: Specify the NamespaceID associated with row in KeyTagAllocationTable
 ▼<Attribute>
    <VendorIdentification type="TextString" value="$TCG-SWG"/>
    <AttributeName type="TextString" value="NamespaceID"/>
    <AttributeValue type="Integer" value="1"/>
  </Attribute>
  <!--  Note: Specify the Key Tag Slot within the specified NamespaceID -->
 ▼<Attribute>
    <VendorIdentification type="TextString" value="$TCG-SWG"/>
    <AttributeName type="TextString" value="KeyTag"/>
    <AttributeValue type="Integer" value="1"/>
  </Attribute>
  <!--  Note: Specify the Key UID of the XTS-AES Key2  -->
 ▼<Link>
    <LinkType type="Enumeration" value="NextLink"/>
    <LinkedObjectIdentifier type="TextString" value="$UNIQUE_IDENTIFIER_MEK_KEY_2"/
  </Link>
 </Attributes>
▼<SymmetricKey>
```

Figure 18: Specifying an MEK and its Attributes with KMIP's Batch Item

Because MEKs are always injected protected using previously provisioned KEKs (or using a Storage Device's Key Per I/O Public Key Certificate (see section 2.2.3.2), an MEKs' Key Block structure needs to always specify the Key Wrapping Data structure (structured in a manner similar to what is shown in the example Figure 17 of encrypted KEK) to convey information about how the key was wrapped.

If the host is specifying both keys for the MEK (which is required if Shared XTS-AES Tweak Key Required is not supported by a TPer (see section 3.1.1.3.10)), the XTS-AES Key2 Batch Item is required to be included in the same Request Message to logically keep the processing of Key1 and Key2 together within the TPer.

In addition, the values specified for the operation and attributes fields for XTS-AES Key2 need to be the same as the values for XTS-AES Key1 as both keys are components of a single MEK that is targeting the same Namespace and the same key tag.

Some attributes' values, however, may differ. For example, XTS-AES Key1 and XTS-AES Key2 may be encrypted by different keys or wrapped using different algorithms, which would result in attributes with different values in the Key Wrapping Data structure. Attributes that uniquely identify the Batch Item and the key like the Batch Item Request ID and the Link Type respectively should also change to identify the Batch Item and the appropriate Link Type value for Key.

Building on the example shown in Figure 18, Figure 19 shows how the Link attribute values change for Key2's Batch Item to link back to Key1.

```
<Link>
  <LinkType type="Enumeration" value="PreviousLink"/>
  <LinkedObjectIdentifier type="TextString" value="$UNIQUE_IDENTIFIER_MEK_KEY_1"/>
</Link>
```

Figure 19: XTS-AES Key2's Link Type Attribute Linking to the KeyUID of the XTS-AES Key1

If a TPer has the Level 0's Shared XTS-AES Tweak Key Required bit set, the host is not expected to declare the Link attribute to point to Key2 (i.e., Tweak Key) as the latter is generated and managed by the Storage Device. If the host specifies it, the TPer will ignore it.

End of informative comment

A TPer SHALL validate that a Batch Item request from the host to inject a MEK is structured as follows:

- The KMIP operation SHALL be an Import operation.
- If multiple MEKs keys are being injected, a unique identifier of each KMIP Batch Item SHALL be declared.
- The KMIP's Message Request Payload SHALL include:
  - The KMIP Unique Identifier of the key being injected with length less than or equal to the maximum value supported by the TPer (see 3.1.1.3.14 for details).
  - The SymmetricKey Object type
  - An Attributes section that includes:
    - A Cryptographic Parameters attribute that includes:
      - The Key Role Type with its value set to DEK
      - A Cryptographic Algorithm with its value set to AES
      - A Cryptographic Length with its value set to 256
        - Optionally, Cryptographic Algorithm and Cryptographic Length attributes MAY be specified inside the Key Block structure outside of the key value.
    - The Vendor Attribute that SHALL include:
      - The Vendor Identification with its value field set to TCG-SWG
      - The Attribute Name and Attribute Value specifying the Namespace ID the TPer SHALL associate the injected key with.
      - The repeated Vendor Identification with its value field set to TCG-SWG
      - The repeated Attribute Name and Attribute Value specifying the Key Tag within the specified Namespace ID that the TPer SHALL associate the injected key with.
      - If the TPer's Key Per I/O Level 0's Shared XTS-AES Tweak Key Required bit is set to zero, Link Attribute with Link Type value SHALL be set to Next Link if the key being imported is XTS-AES Key1. If the key being imported is XTS-AES Key2, Link Type SHALL be set to Previous Link
        - If the TPer's Key Per I/O Level 0's Shared XTS-AES Tweak Key Required bit is set to 1, the Link Attribute SHALL be omitted. If the Link attribute is declared, the TPer SHALL ignore the key2 that the attribute points to.
  - A SymmetricKey structure that SHALL include:

- A Key Block structure that SHALL include:
  - A Key Value Byte String
  - The Key Wrapping Data that, since the injected key is always encrypted, SHALL include:
    - The Wrapping method with its value field set to Encrypt
    - An Encryption Key Information structure that SHALL include:
      - The Unique Identifier of the key previously provisioned into the TPer that was used to wrap the injected key
      - The Cryptographic Parameters that SHALL include:
        - The Cryptographic Algorithm set to AES as the wrapping method used to protect the injected key.
        - The Block Cipher Mode if the wrapping algorithm is AES based, with its value field set to NIST Key Wrap or GCM depending on what the Storage Device supports.
      - An IV/Counter/Nonce if AES-GCM was used to wrap the key
    - An Authenticated Encryption Tag if the injected key is wrapped and AES-GCM was used to wrap the key.

Start of informative comment

The following rule defines how a TPer extracts the Nonce from the ciphered key material if the Replay Protection feature has been enabled by the host on the TPer (see 4.3.5.1.4).

End of informative comment

If Replay Protection has been enabled on a TPer, the TPer SHALL process the ciphered TTLV-encoded MEK key value as follows: Key Material Tag || Byte String || Length || Ciphered Key Material Value, where the first component of the Ciphered Key Material Value (i.e., from its first bit that appears in the sequence up to the specified Cryptographic Length of the key) SHALL represent the actual key data while the remaining bytes represent the Nonce value.

Start of informative comment

The following rule prevents a TPer from processing a KMIP operation to inject an MEK unless KMIP operation, object, and attributes and their values are as specified in section 5.4.4.2.4.

End of informative comment

If a Storage Device receives a KMIP Request Message's Batch Item to import a symmetric key object where DEK Key Role Type is specified and any of the required fields specified in section 5.4.4.2.4 are not specified, the Storage Device SHALL fail the Batch Item operation with Invalid Message Result Reason with the following exception:

If a TPer's Key Per I/O's Level 0 indicates support for Shared Tweak Key Required, then the TPer SHALL process requests to inject an MEK that do not specify the Link attribute.

Start of informative comment

The following rule prevents a TPer from processing a KMIP operation to inject a DEK if the Request Payload specifies attributes that are not applicable to a DEK Key Role Type. For example, specifying a TCG KEK UID in addition to specifying Namespace ID or a Key Tag will result in an error. Or, specifying RSA-OAEP as the encryption method for DEK Key Role Type will also result in an error.

End of informative comment

If a Storage Device receives a KMIP Request Message's Batch Item to import a symmetric key object where a DEK Key Role Type is specified and the Batch Item's payload specifies attributes entries that are only applicable to a different Key Role Type value, the Storage Device SHALL fail the Batch Item operation with Invalid Message Result Reason.

Start of informative comment

The following rule prevents a TPer from processing a KMIP operation for injecting an MEK that specifies a linkage to either of XTS-AES Key1 or XTS-AES Key2 UID unless they are both declared within the same KMIP Request Message.

End of informative comment

If a TPer receives a KMIP Batch Item request to import a symmetric key where:

1. the DEK Key Role Type is specified;
2. the Linked attribute is specified;
3. the operation would otherwise complete successfully; and
4. the TPer's Key Per I/O Level 0's Shared XTS-AES Tweak Key Required bit is set to zero,

then, the KMIP KeyUID specified by the Linked Object Identifier field is expected to belong to a SymmetricKey object of the same Key Role Type, and that object is expected to appear in the same KMIP Request Message that specifies this Batch Item. Otherwise, the TPer SHALL fail the Batch Item operation with Invalid Attribute Value Result Reason.

Start of informative comment

The following rule prevents a TPer from processing KMIP operations for injecting an MEK unless both of the XTS-AES Key1 and Key2 for the same XTS-AES key have the same values for Namespace ID, Key Tag, Cryptographic Algorithm, Cryptographic Length

End of informative comment

If a TPer receives a request to Import symmetric key objects where:

1. the DEK Key Role Type is specified for those objects;
2. the Link Attribute with Previous Link and Next Link values is specified between the objects;
3. the following attributes are specified,
    a. Namespace ID;
    b. Key Tag;
    c. Cryptographic Algorithm;
    d. Cryptographic Length; and
4. the operations would otherwise complete successfully,

Then, the attribute values for those attributes SHALL be the same across those objects. If the values do not match across the linked objects, the TPer SHALL fail the Batch Items' operation with Invalid Attribute Value Result Reason.

Start of informative comment

The following rule prevents a TPer from associating an injected MEK with a Key Tag within a Namespace unless the injected MEK is wrapped with a key included in the allowed list of KEKs for the specified Namespace ID

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

1. the DEK Key Role Type is specified;
2. the AttributeName is specified;
3. the AttributeName value specifies "NamespaceID";
4. the AttributeValue is specified;
5. the specified value of the Unique Identifier field within the Encryption Key Information is associated with a key associated with row in the `KeyEncryptionKey` Table whose UID is not included in the list of UIDs of the AllowedKeyEncryptionKeys column for the `KeyTagAllocation` Table row associated with the specified Namespace ID value; and
6. the operation would otherwise complete successfully,

then, the TPer SHALL fail the Batch Item operation with Permission Denied Result Reason

Start of informative comment

The following rule prevents a TPer from associating an injected MEK with Key Tag within a Namespace if the injected MEK is wrapped with a key that is in an Access Locked state.

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

1. the DEK Key Role Type is specified;
2. the specified value of the Unique Identifier field within the Encryption Key Information is associated with a key associated with a row in `KeyEncryptionKey` Table whose AccessLocked and AccessLockEnabled columns are both set to True;
3. the specified value of the Unique Identifier field within the Encryption Key Information is associated with a key associated with a row in the `KeyEncryptionKey` Table whose UID is included in the list of UIDs of the AllowedKeyEncryptionKeys column for the `KeyTagAllocation` Table row associated with the specified Namespace ID value; and
4. the operation would otherwise complete successfully,

then, the TPer SHALL fail the Batch Item operation with Permission Denied Result Reason

Start of informative comment

The following rule prevents a TPer from associating an injected MEK with Key Tag within a Namespace unless the injected MEK is wrapped with a key whose key value equals the key value stored by the TPer.

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

1. the DEK Key Role Type is specified;
2. the specified value of the Unique Identifier field within the Encryption Key Information is associated with a key associated with a row in `KeyEncryptionKey` Table whose AccessLocked or AccessLockEnabled column is set to False;
3. the specified value of the Unique Identifier field within the Encryption Key Information is associated with a key associated with row in the `KeyEncryptionKey` Table whose UID is included in the list of UIDs of the

AllowedKeyEncryptionKeys column for the KeyTagAllocation Table row associated with the specified Namespace ID value;

4. the operation would otherwise complete successfully; and
5. unwrapping fails,

then, the TPer SHALL fail the Batch Item operation with Cryptographic Failure Result Reason

Start of informative comment

The following rule prevents a TPer from associating an injected MEK with Key Tag within a Namespace unless the injected MEK is wrapped with a key that has already been injected into the TPer.

End of informative comment

If a TPer receives a KMIP Batch Item's Request Payload to import a symmetric key where:

4. the DEK Key Role Type is specified;
5. the specified value of the Unique Identifier field within the Encryption Key Information is not associated with a key associated with a row in KeyEncryptionKey Table; and
6. the operation would otherwise complete successfully,

then, the TPer SHALL fail the Batch Item operation with Invalid Attribute Result Reason.

5.4.4.2.5 Response Message Considerations

5.4.4.2.5.1 Response Message Header

Start of informative comment

```
▼<ResponseMessage>
  ▼<ResponseHeader>
     <!--  Note: Protocol version required  -->
    ▼<ProtocolVersion>
       <ProtocolVersionMajor type="Integer" value="2"/>
       <ProtocolVersionMinor type="Integer" value="1"/>
     </ProtocolVersion>
     <!--  TimeStamp is required  -->
     <!--  0 indicates real-time clock not supported (refer to TCG core 3.3.4.7.4)  -->
     <TimeStamp type="DateTime" value="0"/>
     <BatchCount type="Integer" value="2"/>
   </ResponseHeader>
```

Figure 20: Response Message Header Structure

Figure 20 shows a typical KMIP Response Message header with its required fields for a Key Per I/O KMIP request.

All response messages generally use a similar header structure as shown in the figure.

The response header indicates the supported Protocol Version, Time Stamp, and Batch Count fields if multiple Batch Items were included in the request message for which the response is prepared.

The supported Protocol Version must be 2.0 or greater.

The Time Stamp field could be used by a TPer to time stamp the response to the host if the TPer supports real time clock capability. For Storage Devices that do not support real time clock capabilities, a value of 0 must be returned to align to TCG's expected response for such Storage Devices (see section 3.3.4.7.4. of [3])

The Batch Count field is the number of the Batch Items included in the response. If the request for which the response is prepared included multiple Batch Items, the response message must include the responses for all those Batch Items

batched (i.e., synchronous processing). Asynchronous processing of Batch Items is outside the scope of this specification.

End of informative comment

If a TPer does not support a real time clock capability, it SHALL return all 0s in the value field of Time Stamp.

5.4.4.2.5.2 Batch Item Response

Start of informative comment

```
<ResponseMessage>
  <ResponseHeader>
    <!--  Note: Protocol version required  -->
    <ProtocolVersion>
      <ProtocolVersionMajor type="Integer" value="2"/>
      <ProtocolVersionMinor type="Integer" value="1"/>
    </ProtocolVersion>
    <!--  TimeStamp is required  -->
    <!--  0 indicates real-time clock not supported (refer to TCG core 3.3.4.7.4)  -->
    <TimeStamp type="DateTime" value="0"/>
    <BatchCount type="Integer" value="2"/>
  </ResponseHeader>
  <!--  Batch Item 1: Response to Inject AES-XTS MEK Key1  -->
  <BatchItem>
    <Operation type="Enumeration" value="Import"/>
    <!--  UniqueBatchItemID value shall match value in request  -->
    <UniqueBatchItemID type="ByteString" value="01"/>
    <ResultStatus type="Enumeration" value="Success"/>
    <ResponsePayload>
      <!--  UniqueIdentifier value shall match value in request  -->
      <UniqueIdentifier type="TextString" value="$UNIQUE_IDENTIFIER_MEK_KEY_1"/>
    </ResponsePayload>
  </BatchItem>
  <!--  Batch Item 2: Response to Inject AES-XTS MEK Key2  -->
  <BatchItem>
    <Operation type="Enumeration" value="Import"/>
    <!--  UniqueBatchItemID value shall match value in request  -->
    <UniqueBatchItemID type="ByteString" value="02"/>
    <ResultStatus type="Enumeration" value="Success"/>
    <ResponsePayload>
      <!--  UniqueIdentifier value shall match value in request  -->
      <UniqueIdentifier type="TextString" value="$UNIQUE_IDENTIFIER_MEK_KEY_2"/>
    </ResponsePayload>
  </BatchItem>
</ResponseMessage>
```

Figure 21: Response Message Batch Items

Figure 21 gives an example of Batch Items responses.

As the figure shows, Batch Items responses are expected to be batched if the request message for which the response is prepared included multiple Batch Items.

When multiple Batch Items are included, a Unique Batch Item ID of each Batch Item needs to be specified and its value must match the value that was specified by the host in the Request Message for which the response is intended.

As specified by [9], the response Batch Item must also specify the operation that caused the request, the Result Status of the Operation, the Result Reason of the failure if a failure has occurred, and the Response Payload.

End of informative comment

5.4.4.3 Discovering Supported KMIP Capabilities

5.4.4.3.1 Supported Versions

A Key Per I/O SSC compliant Storage Device SHALL implement the Discover Versions operation as defined in [9] to allow the host to discover which KMIP protocol versions the Storage Device supports. The Protocol Version returned in the operation response payload SHALL be 2.0 or greater.

5.4.4.3.2 Supported KMIP Operations and Object Types

A Key Per I/O SSC compliant Storage Device SHALL implement the Query operation as defined in section 5.4.2.3 to allow the host to discover KMIP Operations and KMIP Object Types that the Storage Device support.

The list of operations returned by a Key Per I/O SSC compliant Storage Device in the Query response payload SHALL include the Query operation itself, in addition to Import and Discover Versions operations. For Object Types supported, the Storage Device SHALL return a response payload that includes the "Symmetric Key" object type.

5.5 Interface Read and Write Interactions

5.5.1 Reading User Data

When the host initiates an Interface Read command specifying a Key Tag that is associated with a valid MEK, then the Storage Device SHALL perform the read operation and decrypt user data using the MEK associated with the specified Key Tag.

If the Storage Device receives an Interface Read command specifying a Key Tag that does not have a valid MEK associated with it, then the Interface Read command SHALL fail with a TCG status of Invalid Key (see [5]).

Start of informative comment

For usability, it is desirable for a Storage Device to differentiate during read operations between incorrect key usage and data integrity issues. Because of this, a Storage Device may track MEK usage on Interface Write commands in order to detect when an Interface Read command will fail because the incorrect MEK was used to read the data stored at the read location.

Support for this capability is identified via the Incorrect Key Detection Supported bit in the Key Per I/O SSC Feature Descriptor (see section 3.1.1.3.11).

End of informative comment

Depending on the Incorrect Key Detection Supported bit in the Key Per I/O SSC Feature Descriptor, the Storage Device may handle Interface Read commands differently as follows:

If the Incorrect Key Detection Supported bit is set to one, and the Storage Device receives an Interface Read command that would result in the Storage Device using a different MEK to access the user data than the MEK that the user data was originally written with, for one or more LBAs specified in that Read command, then the Storage Device SHALL fail the Interface Read command with a status of Incorrect Decryption Key (see [5]).

Otherwise, if the Incorrect Key Detection Supported bit is cleared to zero, and the Storage Device receives an Interface Read command that would result in the Storage Device using a different MEK to access the user data than the MEK that the user data was originally written with, then the behavior of a TPer is vendor specific.

If the Storage Device receives an Interface Read command specifying a Key Tag for a Namespace which is not managed by the Key Per I/O SP, then the Interface Read command SHALL fail with a status of Other Invalid Command Parameter (see [5]).

5.5.2 Writing User Data

When the host initiates an Interface Write command specifying a Key Tag that is associated with a valid MEK, then the Storage Device SHALL perform the write operation and encrypt user data using the MEK associated with the specified Key Tag.

If the Storage Device receives an Interface Write command specifying a Key Tag that does not have a valid MEK associated with it, then the Interface Write command SHALL fail with a status of Invalid Key (see [5]).

If the Storage Device receives an Interface Write command specifying a Key Tag for a Namespace that is not managed by the Key Per I/O SP, then the Interface Write command SHALL fail with a status of Other Invalid Command Parameter (see [5]).

5.6 Interactions with Resets

5.6.1 Interactions with Power Cycle

Media Encryption Keys injected via the Media Encryption Key Injection command SHALL NOT be stored in the Storage Device in non-volatile storage media.

If the Key Per I/O SP is owned, all Media Encryption Keys injected via the KMIP interface SHALL be cleared from volatile storage media as a result of a TCG Power Cycle Reset.

Other TCG resets SHALL NOT cause Media Encryption Keys to be cleared from volatile storage media.

5.7 Interactions with NVMe Namespace Management

An NVM subsystem MAY support the Namespace Management command.

If the Key Per I/O SP is owned, then execution of the Namespace Management command with the Select (SEL) field set to Delete on a Namespace that is managed by Key Per I/O SHALL fail with a status of Invalid Security State (see [5]).

5.8 Interactions with the Format NVM Command

The Format NVM command MAY be supported on an NVM subsystem that contains the Key Per I/O SP.

If the Key Per I/O SP is in a Manufactured state, then the Format NVM command with the Secure Erase Settings field set to Cryptographic Erase issued on a Namespace that is managed by Key Per I/O SHALL fail with a status of Invalid Security State (see [5]).

5.9 Interactions with the Sanitize Command

The Sanitize command MAY be supported on an NVM subsystem that contains the Key Per I/O SP.

If the Key Per I/O SP is owned, then the Storage Device SHALL:

a) Report that the Sanitize command is not supported (i.e., the SANICAP field is zero); or

b) Report that the Sanitize command is supported (i.e., the SANICAP field is non-zero) and terminate the Sanitize command with a Data Protection Error (see [5]).

5.10 Interactions with the Locking Template

If a TPer supports the Locking SP and the Key Per I/O SP, then the TPer SHALL support the Manufactured-Inactive state for the Locking SP.

If the Locking SP is owned, then the TPer SHALL prohibit the Key Per I/O SP from transitioning out of the Manufactured-Inactive state.

If the lifecycle state of the Key Per I/O SP is in the Manufactured state, then the TPer SHALL prohibit the Locking SP from transitioning out of the Manufactured-Inactive state.

If the Locking SP is owned, then invocation of the `Activate` method on the Key Per I/O SP SHALL fail with a status of FAIL.

If the Key Per I/O SP is owned, then invocation of the `Activate` method on the Locking SP SHALL fail with a status of FAIL.

5.11 Interactions with the Identify Namespace data structure

If a TPer supports management of a Namespace via the Key Per I/O SP, then the TPer SHALL set the KPIOSNS bit of the Identify Namespace data structure for that Namespace (see [15]) to one.

If the TPer does not support management of a Namespace via the Key Per I/O SP, then the TPer SHALL clear the KPIOSNS bit of the Identify Namespace data structure for that Namespace to zero.

If the Key Per I/O SP has been configured to manage a Namespace (see section 4.3.5.2.5), then the TPer SHALL set the KPIOENS bit of the Identify Namespace data structure for that Namespace to one.

If the Key Per I/O SP has been configured to not manage a Namespace (see section 4.3.5.2.5), then the TPer SHALL clear the KPIOENS bit of the Identify Namespace data structure for that Namespace to zero.

5.12 Interactions with the Identify Controller data structure

Start of informative comment

The Key Per I/O Scope (KPIOSC) bit of the Key Per I/O Capabilities field in NVMe's Identify Controller data structure (see [15]) indicates whether the Key Per I/O capability applies to all Namespaces in the NVM subsystem. This bit has interactions with the Managed column of the `KeyTagAllocation` Table in the Key Per I/O SP.

End of informative comment

If the Key Per I/O Scope bit of the Key Per I/O Capabilities field in the Identify Controller data structure is set to one, then the Managed column of all rows in the `KeyTagAllocation` Table SHALL default to a value of one.

If the Key Per I/O Scope bit of the Key Per I/O Capabilities field in the Identify Controller data structure is cleared to zero, then the Managed column of all rows in the `KeyTagAllocation` Table SHALL default to a value of zero.

If the Key Per I/O Scope bit of the Key Per I/O Capabilities field in the Identify Controller data structure is set to one, then invocation of the `Activate` method on the Key Per I/O SP SHALL cause all existing user data in the NVM subsystem to no longer be recoverable. The mechanism for how the Storage Device makes user data not recoverable is implementation specific.

If the Key Per I/O Scope bit of the Key Per I/O Capabilities field in the Identify Controller data structure is cleared to zero, then invocation of the `Activate` method on the Key Per I/O SP SHALL NOT cause existing user data in the NVM subsystem to no longer be recoverable.

If the Key Per I/O Scope bit of the Key Per I/O Capabilities field in the Identify Controller data structure is set to one, and the `Set` method is invoked on any row in the `KeyTagAllocation` Table with the Managed column specified, then the `Set` method SHALL fail with a status of INVALID_PARAMETER.

