TCG Storage
Security Subsystem Class: Enterprise

Specification Version 1.00 Final

Revision 3.00

January 10, 2011

Contacts:

admin@trustedcomputinggroup.org

TCG

Revision History

Version 1.00	Date	Description
Rev 1.00	27 January, 2009	First publication
Rev 2.00	21 December, 2009	Revised document to incorporate informative appendices into document body to improve readability.
Rev 3.00	10 January, 2011	Incorporated Authenticate Method Failures section, corrected reference in Properties Method Deviations section, added AUTHORITY_LOCKED_OUT section, added MakerSymK authority, clarifications to read/write data and ReadLocked/WriteLocked, fixed page numbering issues and table page break issues, inconsistencies in markup/other editorial issues.

TABLE OF CONTENTS

Tables

1 Introduction

The Enterprise SSC is based on a draft Core Specification [2] identified as not for implementation. The Enterprise SSC incorporates deviations to that specification in order to enable implementation of this SSC.

1.1 Document Purpose

The Storage Workgroup specifications are intended to provide a comprehensive architecture for putting storage devices under policy control as determined by the trusted platform host, the capabilities of the storage device to conform with the policies of the trusted platform, and the lifecycle state of the storage device as a Trusted Peripheral.

1.2 Security Subsystem Classes

[2] defines the TCG-related functions for a TCG Trusted Storage Device. However, not all trusted storage devices might support all functionality. There are multiple “classes” of compliance to [2], called Security Subsystem Classes (SSCs).

Security Subsystem Classes explicitly define the minimum acceptable capabilities of a storage device in a specific “class”. A storage device in a specific class MAY have only some of the capabilities (tables, methods, access controls) defined in [2] and MAY include additional capabilities definitions.

1.3 Scope and Intended Audience

This SSC specification is an implementation profile for storage devices built to:

- protect the confidentiality of stored user data and
- minimize the time to bring devices online.

A single threat model is assumed: unauthorized access to user data on the device once it leaves the owner’s control. The specification’s scope is storage devices deployed in systems that implement Fibre Channel (FC), Serial Attached SCSI (SAS), and Serial ATA (SATA) interfaces.

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

1.4 Goals

The goal of this specification is to define an implementation profile for storage devices that ensures interoperability between different vendor solutions. This is achieved by:

- Identification of a minimum subset of required functionality from [2];
- Definition of additional functionality needed to satisfy enterprise-class storage use cases;
- Definition of expected storage device behavior for all required functionality.

1.5 Key Words

The key words "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", and "MAY" in this specification are to be interpreted as described in [1]. These keywords are capitalized when used to unambiguously specify requirements over protocol and features behavior that affect the interoperability and security of the implementation. When these words are not capitalized, they are meant in their natural-language sense.

Additionally, the following terms are used in this specification to describe the requirement of particular features, including tables, methods, and usages thereof.

- **Mandatory (M):** The feature SHALL be supported by the storage device in order to be compliant with this specification. A compliance test SHALL validate the feature is operational.

- **Optional (O):** The feature MAY be supported by the storage device. If implemented, a compliance test SHALL validate the feature is operational.

1.6 Precedence

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

1) this specification;

2) the Storage Interface Interactions Specification (see[7]); and

4) the Core Specification (see [2]).

1.7 References

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

[2]. Trusted Computing Group (TCG), 2007, “TCG Storage Architecture Core Specification”, Version 1.0, Revision 0.9 – Draft.

[3]. NIST, “Computer Security Division, Cryptographic Toolkit”, http://csrc.nist.gov

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

[5]. [INCITS T10/1731-D], “Information technology - SCSI Primary Commands - 4 (SPC-4)”

[6]. [ANSI INCITS 452-2008], “Information technology - AT Attachment 8 - ATA/ATAPI Command Set (ATA8-ACS)”

[7]. Trusted Computing Group (TCG), “TCG Storage Interface Interactions Specification”, Version 1.0, Revision 1.0

1.8 Definition of Terms

Term	Definition
IF-RECV	An interface command used to retrieve security protocol data from the TPer (see [2]).
IF-SEND	An interface command used to transmit security protocol data to the TPer (see [2]).
Locking SP	A security provider that incorporates the Locking Template. The Locking Template is defined in [2].
Read command	See: Read user data.
Read user data	an operation requested by the host to transfer user data to the host
SSC	Security Subsystem Class (see [2]) specifications describe profiled sets of TCG functionality
TCG Reset	A high-level reset type defined in [2].
TPer	The TCG security subsystem within a storage device (see [2]).
User data	data that may be transferred between the host and the device using read commands and write commands

Term	Definition
VU	Parameters that are vendor unique
Write command	See: Write user data
Write user data	an operation requested by the host to modify user data, that may include transferring data from the application client to the device

2 Overview

Begin Informative Content

This specification is an implementation profile for trusted storage devices commonly deployed within Enterprise-class systems. It provides storage device implementation requirements needed to guarantee interoperability between storage devices from different vendors. Enterprise-class systems often deploy a mix of cross-vendor storage devices and interoperability is therefore key, both for non-trusted and trusted storage devices.

This specification defines a limited set of TCG Trusted Storage functionality that, combined with Full Disk Encryption (FDE), protects the confidentiality of user data at rest. Only a single threat scenario is addressed: removal of the storage device from its host system involving a power cycle of the storage device and subsequent unauthorized access to data stored on that device.

This specification assumes that hosts in Enterprise systems could have limited (computational) capabilities and/or operate within a system that has strict response time requirements. Based on this assumption, the objective of this specification is to define strict boundaries on the host-device communication protocol and data structures used in the TCG Storage Architecture. This prevents the host from having to maintain security configuration information on a per storage device basis and allows it to expect similar behavior for each SSC compliant storage device within the system.

To avoid requiring that the host performs dynamic discovery of features and values, the storage device behavior is unambiguously defined, and as such creation and deletion of tables and/or rows within tables is not required. This specification defines 2 SPs, the tables that are host-accessible within each SP, and the values within each table. The SPs and tables MAY be present in the storage device when it leaves manufacturing, See section 11.1.

This specification addresses a limited set of use scenarios. These scenarios are:

- Deploy storage device & Take Ownership: the device is integrated into its target system and ownership transferred by actively setting or changing the device's owner credential.
- Activate or Enroll Device: LBA ranges are configured, data encryption and access control credentials (re)generated and/or set on the storage device.
- Lock & Unlock Device: active unlocking of one or more LBA ranges by the host and locking of those ranges under host control via either an explicit lock or implicit lock triggered by a reset event.
- Repurpose & End-of-Life: erasure of data within one or more LBA ranges and reset of locking credential(s) for storage device repurposing or decommissioning.

End Informative Content

3 SSC Features and Capability Definitions

3.1 Interface Communications Protocol

An Enterprise SSC-compliant storage device SHALL implement the synchronous communications protocol (see section 4.4) using the SCSI (T10) or ATA (T13) defined security protocol commands. This SSC's implementation of the synchronous communications protocol calls for a single ComPacket / Packet / Subpacket combination per interface command and defines two Active ComIDs for communications using ComPackets.

3.2 Cryptographic Features

The storage device SHALL implement Full Disk Encryption for all host accessible user data stored on media. The storage device SHALL support AES 128 or AES 256 (see [4]).

3.3 Authentication

The storage device SHALL support password authorities and authentication with a maximum credential password size of 32-bytes.

3.4 Table Management

The tables and table rows required by this specification MAY be present in the storage device when the device leaves the manufacturer. The creation or deletion of tables in SPs post-manufacturing is outside the scope of this specification. The creation or deletion of rows in tables post-manufacturing is outside the scope of this specification.

3.5 Issuance

The SPs required by this specification MAY be present in the storage device when the storage device leaves the manufacturer. The issuance of SPs post-manufacturing is outside the scope of this specification.

3.6 SSC Discovery

Discovery is a process for the Host to examine the storage device's configurations and capabilities.

3.6.1 Discovery levels

Discovery is a process used to determine the capabilities of the TPer.

- **Level 0:** This discovery request is sent as an IF-RECV command. The Security Protocol SHALL be `0x01` and the ComID SHALL be `0x0001`. The TPer SHALL support the requirements in 3.6.2.
- **Level 1:** These TCG methods request basic TPer capabilities (i.e. via `Properties`) using simple host messaging requirements. The required support is defined in this specification.
- **Level 2:** TCG methods retrieve specified table cell values. The required support is defined in this specification.

3.6.2 Level 0 Discovery

This section identifies deviations from [2] that are required by this specification. The TPer SHALL support the LEVEL 0 DISCOVERY response data format described in this section.

The LEVEL 0 DISCOVERY command provides a host with some basic information about TPer capabilities; both current and potential. More detailed information is obtainable through SP operations.

3.6.2.1 IF-SEND Command

IF-SEND command, with

Security Protocol = 01h

Security Protocol Specific = 0001h

Transfer Length= n/a

There is no IF-SEND command defined for Level 0 Discovery. The TPer SHALL transfer all of the data from the host, SHALL discard it, and return 'good' status to the host.

3.6.2.2 IF-RECV Command

IF-RECV command, with

Security Protocol = 01h

Security Protocol Specific = 0001h

Allocation Length = maximum length of the LEVEL 0 DISCOVERY response data that the host elects to receive.

This IF-RECV command MAY be processed at any time, without regard to sessions or prior authentication.

If the Allocation Length is less than the size of the LEVEL 0 DISCOVERY response data that is available, the TPer SHALL return the requested amount of data, even if it is truncated.

If the Allocation Length is greater than the size of the LEVEL 0 DISCOVERY response data:

a) An ATA device SHALL pad with zeros to the Allocation Length requested.

b) A SCSI target with INC_512 set to one SHALL pad with zeroes to the next 512-byte boundary. If INC_512 is set to zero, the target SHALL only transfer the available number of bytes.

The LEVEL 0 DISCOVERY response data (see Table 01) consists of a header field (see Table 02) and zero or more variable length feature descriptors (see 3.6.2.3). A TPer SHALL not include feature descriptors for features that it does not implement. The data does not contain any ComPackets, and is not contained within a ComPacket.

Table 01 Level 0 DISCOVERY response data format

Byte \ Bit	7	6	5	4	3	2	1	0
0 – 47	Level 0 Discovery header (see Table 02)							
48 – n	Feature Descriptor(s)							

Table 02 Level 0 Discovery header

Byte \ Bit	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)

3.6.2.2.1 Length of parameter data

Indicates the total number of bytes that are valid in the Level 0 Discovery header and all of the feature descriptors returned, not including this field.

3.6.2.2.2 Data structure version number

This version number describes the format of the Level 0 Discovery header returned. The value SHALL be `00000001h`

3.6.2.2.3 Vendor Specific

These bytes are vendor specific.

3.6.2.3 Features - Overview

A feature is a set of capabilities that MAY be implemented in a TPer. A Host MAY discover the capabilities and properties of a TPer by examining its feature descriptors. Features that are implemented by a TPer SHALL be indicated by the presence of a feature descriptor.

The feature descriptors SHALL be returned in the LEVEL 0 DISCOVERY response data in order of increasing feature code values. Features that are not implemented SHALL NOT be returned.

All feature descriptors SHALL conform to the general format defined in Table 03.

Table 03 Feature Descriptor template format

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)			Feature Code				
1								(LSB)
2	Version					Reserved		
3	Length							
4 – n	Feature Dependent Data							

3.6.2.3.1 Feature Code

The Feature Code field SHALL identify a feature implemented by the TPer.

3.6.2.3.2 Version

The Version field describes the format of the data returned. Future versions of a feature SHOULD be backward compatible; incompatible changes SHOULD be included in a different feature.

3.6.2.3.3 Length

The Length field indicates the length of the Feature Dependent Data (in bytes) that follow this header. This field SHALL be an integral multiple of 4.

This information reports support for various TPer parameters. This mandatory feature SHALL always be returned in the Level 0 Discovery response.

These parameters indicate whether the TPer supports a variety of features. Having a given “support’ flag true does not imply that the feature is required or enabled. Actually enabling a feature MAY require personalization of the TPer.

3.6.2.4 TPer feature

This information reports support for various TPer parameters. This mandatory feature SHALL always be returned in the Level 0 Discovery response.

These parameters indicate whether the TPer supports a variety of features. Having a given “support’ flag true does not imply that the feature is required or enabled. Actually enabling a feature MAY require personalization of the TPer.

Table 04 TPer feature

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

3.6.2.4.1 SyncSupported

SyncSupported SHALL be set to one if the TPer supports the Synchronous Protocol, otherwise SyncSupported SHALL be cleared to zero.

3.6.2.4.2 AsynchSupported

AsynchSupported SHALL be set to one if the TPer supports the Asynchronous Protocol, otherwise AsynchSupported SHALL be cleared to zero.

3.6.2.4.3 ACK/NAKSupported

ACK/NAKSupported SHALL be set to one if the TPer supports transmission ACK/NAK flow control for communications, otherwise ACK/NAKSupported SHALL be cleared to zero.

3.6.2.4.4 BufferMgmtSupported

BufferMgmtSupported SHALL be set to one if the TPer supports buffer management flow control for communications, otherwise BufferMgmtSupported SHALL be cleared to zero.

3.6.2.4.5 StreamingSupported

StreamingSupported SHALL be set to one if the TPer supports the streaming protocol, otherwise StreamingSupported SHALL be cleared to zero.

3.6.2.4.6 ComID Management Supported

ComIDManagementSupported SHALL be set to one if the TPer supports ComID management using Protocol ID `02h`, otherwise SHALL be cleared to zero.

3.6.2.4.7 Required Values

The TPer SHALL return the TPer Feature in the Level 0 Discovery response with:

Feature Code = `0x0001`

Version = `0x1` or any version that supports the defined features in this SSC.

Length = `0x0C`

SyncSupported = 1
StreamingSupported = 1

3.6.2.5 Locking Feature

This information indicates support for an issued Locking template. This mandatory feature SHALL always be returned in the Level 0 Discovery response.

Table 05 Locking feature descriptor

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)			Feature Code				
1								(LSB)
2	Version					Reserved		
3	Length							
4	Reserved	MBR Done	MBR Enabled	Media Encryption	Locked	Locking Enabled	Locking Supported	
5 - 15	Reserved							

3.6.2.5.1 LockingSupported

LockingSupported SHALL be set to one if the TPer supports the Locking template; otherwise

LockingSupported SHALL be set to zero.

3.6.2.5.2 LockingEnabled

LockingEnabled SHALL be set to one if an SP that incorporates the Locking template is in any state other than nonexistent or manufactured-inactive; otherwise LockingEnabled SHALL be set to zero.

3.6.2.5.3 Locked

Locked SHALL be set to one if LockingEnabled is set to one, and one or more LBA ranges in the `Locking` table have either (`ReadLockEnabled`=True and `ReadLocked`=True) or (`WriteLockEnabled`=True and `WriteLocked`=True); otherwise Locked SHALL be set to zero.

3.6.2.5.4 MediaEncryption

MediaEncryption SHALL be set to one if the TPer supports media encryption; otherwise MediaEncryption SHALL be set to zero.

3.6.2.5.5 MBREnabled

MBREnabled SHALL be set to one if LockingEnabled is set to one, and the `MBRControl` and `MBR` tables are implemented, and the `MBRControl` table's `Enabled` column has a value of "True"; otherwise MBREnabled SHALL be set to zero.

3.6.2.5.6 MBRDone

MBRDone SHALL be set to one if MBREnabled is set to one, and the `MBRControl` table's `Done` column has a value of "True"; otherwise MBRDone SHALL be set to zero.

3.6.2.5.7 Required Values

The TPer SHALL return the Locking Feature in the Level 0 Discovery response with:

The Feature Code	= `0x0002`
Version	= `0x1` or any version that supports the defined features in this SSC.
Length	= `0x0C`
LockingSupported	= 1
MediaEncryption	= 1

3.6.2.6 Common SSC feature information

This information is supplied as part of every reported SSC feature.

Table 06 Common SSC Information

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)			Base ComID				
1								(LSB)
2	(MSB)			Number of ComIDs				
3								(LSB)
4 - 15	Reserved for future common SSC parameters							

3.6.2.6.1 Base ComID

This is the lowest static, pre-assigned ComID that the SSC supports for Protocol ID=`01h` sessions.

3.6.2.6.2 Number of ComIDs

This specifies the number of static, pre-assigned ComIDs that the SSC supports for Protocol ID=`01h` sessions, starting at the Base ComID.

3.6.2.7 Enterprise SSC Feature

The TPer SHALL return the Enterprise feature in the Level 0 Discovery response.

Table 07 Enterprise SSC Descriptor Format

Byte \ Bit	7	6	5	4	3	2	1	0
0	(MSB)			Feature Code				
1								(LSB)
2	Version				Reserved			
3	Length							
4	(MSB)			Base ComID				
5								(LSB)
6	(MSB)			Number of ComIDs				
7								(LSB)
8	Reserved for future common SSC parameters							Range Crossing
9 - 19	Reserved for future common SSC parameters							

The TPer SHALL support:

Feature Code = `0x0100`

Version = `0x1` or any version that supports the defined features in this SSC.

Length = `0x10`

Base ComID = base ComID supported by the TPer (e.g., `0x07FE`)

Number of ComIDs = `0x0002` min

Range Crossing = VU 0 = The Storage device supports commands addressing consecutive LBAs in more than one LBA range if all the LBA ranges addressed are unlocked (see 11.4.10).

1 = The storage device terminates commands addressing consecutive LBAs in more than one LBA range (see 11.4.10).

4 Communications

4.1 Communication Properties

The TPer SHALL support fixed size communication buffers with a minimum size of 1024 bytes each. The number of buffers SHALL be sufficient to support concurrent communications with each ComID and/or open session as supported by the TPer, whichever is greater.

For each ComID, the physical size of the buffers SHALL be reported to the host via the `Properties` method (see section 9.2.2.1).

The TPer SHALL terminate any IF-SEND command whose transfer length is greater than the reported MaxComPacketSize size for the corresponding ComID. For details, reference “Invalid Transfer Length parameter on IF-SEND” in [7].

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.

4.2 Supported Security Protocols

The TPer SHALL support IF-RECV commands with a Security Protocol value of `0x00`, `0x01` and `0x02`. The TPer SHALL support IF-SEND commands with a Security Protocol value of `0x01` and `0x02`.

If the host sends an IF-SEND or IF-RECV to an unsupported Security Protocol, the TPer SHALL terminate the command with “Invalid Security Protocol ID parameter” as defined in [7].

For an IF-RECV command with Security Protocol set to `0x00` and Security Protocol Specific set to `0x0000` (Return list of supported protocols), the TPer SHALL respond in accordance to the SCSI (see [5]) or ATA (see [6]) specifications for Security Protocol In and Trusted Receive.

For an IF-RECV command with Security Protocol set to `0x00` and Security Protocol Specific set to `0x0001` (Return a certificate), the TPer SHALL respond in accordance to the SCSI (see [5]) or ATA (see [6]) specifications for Security Protocol In and Trusted Receive.

For an IF-RECV command with Security Protocol set to `0x01` and Security Protocol Specific set to `0x0001` (Level 0 Discovery), the TPer SHALL return one or more 512-byte blocks that describe the attributes of the TCG security protocol corresponding to Security Protocol `0x01`. The return data structure SHALL comply with the requirements in 3.6.2.

The TPer SHALL support IF-SEND and IF-RECV commands with Security Protocol set to `0x02` for the Protocol Stack Reset Command function defined in 4.5.2. If the host sends an IF-SEND with Security Protocol set to `0x02` and an invalid or unsupported Request code in the payload, the TPer SHALL prepare a response with "No Response Available" as defined in 4.5.2.

4.3 ComIDs

- For the purpose of communication using Security Protocol 0x01, the TPer SHALL support:
- ComIDs values 0x07FE and 0x07FF and Extended ComID values 0x07FE0000 and 0x07FF0000 for communications using the Synchronous Protocol, and
- ComID value 0x0001 for Device Level 0 Discovery. See section 3.6 and 3.6.2.

Mandatory ComIDs SHALL be Active (in the “Issued” or “Associated” state).

4.3.1 Inactive or Unsupported ComIDs

If the host uses an inactive or unsupported ComID in an IF-SEND or IF-RECV, the TPer SHALL respond as defined in this section. This section identifies deviations from definition of ComID handling in [2] that are required by this specification.

4.3.1.1 IF-SEND

If the host sends an IF-SEND command to the TPer with a ComID value in the non-reserved range (1000h – FFFFh), and the ComID is in the Inactive state:

• If the TPer supports dynamic ComID allocation, the TPer SHALL:

o Accept all data in the payload of the IF-SEND command and complete the command normally with good status (provided there are no other errors which would cause the command to abort at the interface level)

o Ignore and discard the entire payload of the IF-SEND command.

• If the TPer does not support dynamic ComID allocation, the TPer SHALL:

o Report “Other Invalid CDB parameter” (as specified in [7])

OR

o Perform the action described above for TPers that support dynamic ComID allocation

If the host sends an IF-SEND command to the TPer with a ComID value in the reserved range (0000h – 0FFFh), and the ComID is not supported by the TPer, the TPer SHALL:

• Report “Other Invalid CDB parameter” (as specified in [7])

4.3.1.2 IF-RECV

If the host sends an IF-RECV command to the TPer with a ComID value in the non-reserved range (1000h – FFFFh), and the ComID is in the Inactive state:

• If the TPer supports dynamic ComID allocation, the TPer SHALL:

o Respond to the IF-RECV with a zero-length ComPacket (a ComPacket header only) in the IF-RECV payload. The fields in the ComPacket header SHALL contain:

- **ExtendedComID =** {<ComID from SP_Specific field of CDB>, FFFFh}

  • Note: The value of FFFFh in bits 15 through 0 of the **ExtendedComID** field is an indication to the host that the ComID it is attempting to use is inactive, and that it should not expect to receive any data on that ComID.

- **OutstandingData =** 00000000h
- **MinTransfer =** 00000000h

- **Length =** 00000000h

o Complete the command normally with good status (provided there are no other errors which would cause the command to abort at the interface level)

• If the TPer does not support dynamic ComID allocation, the TPer SHALL:

o Report “Other Invalid CDB parameter” (as specified in [7])

OR

o Perform the action described above for TPers that support dynamic ComID allocation

If the host sends an IF-RECV command to the TPer with a ComID value in the reserved range (0000h – 0FFFh), and the ComID is not supported by the TPer, the TPer SHALL:

• Report “Other Invalid CDB parameter” (as specified in [7])

4.3.1.3 Reserved Values for Extended ComIDs

The value of FFFFh in bits 15 through 0 of the ExtendedComID field is reserved to indicate that the host has attempted to communicate using an inactive ComID.

When assigning Extended ComIDs via the **GET_COMID** command on Protocol ID 02h, the TPer SHALL NOT assign the value of FFFFh in bits 15 through 0 of the ExtendedComID field.

4.4 Synchronous Protocol

The TPer SHALL support the Synchronous Protocol for communications using ComPackets.

Begin Informative Text

The communications protocol stack as described in [2] enables a fully asynchronous exchange of data between host and TPer. Using the communications stack in this manner is a matter of arbitrarily interleaving IF-SEND commands with IF-RECV commands.

Asynchronous communications allows the host to transmit methods and data to the TPer without having to retrieve the results of those methods before sending additional methods; and enables the TPer to return method results, upon request, at arbitrary boundaries. Flow control provides a mechanism for buffer management to occur as data is successfully transmitted and received.

However, for some hosts and devices, these mechanisms are more complex and require more processing capability and code space than MAY be realistically available. For these situations, the communications protocol stack MAY be tailored to better meet the capabilities of the TPer.

For instance, fixed or semi-fixed sized commands simplifies message creation and parsing; and fixed buffer sizes along with restrictions on the relationship between IF-SEND and IF-RECV negates the need for the communications to require flow control for buffer management.

End Informative Text

4.4.1 Protocol States and State Transitions

Figure 1 describes the synchronous protocol states and state transitions.

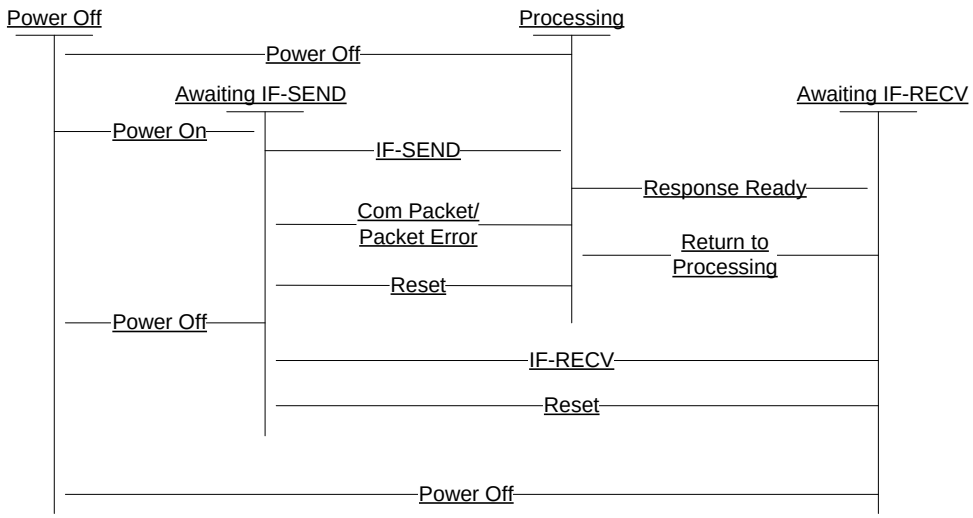

Figure 1– Synchronous Protocol Stack State Diagram

This specification defines the following protocol states for each valid ComID:

State "Power-Off" – In this state, power is removed from the TPer and it is completely unresponsive.

State "Awaiting IF-SEND" – In this state, the TPer command interface is ready and there are no outstanding IF-SEND/IF-RECV commands for the specified ComID. A command is "outstanding" if it has entered the "Processing" or "Awaiting IF-RECV" state. A command is not considered "outstanding" if it is sitting in the TPer command queue awaiting initial processing by the device.

While in this state, if IF-SEND is received or dequeued with the ComID for this state machine, the TPer MAY request command payload transfer and SHALL return interface status to the host.

While in this state, if IF-RECV is received or dequeued with the ComID for this state machine, the TPer SHALL return a response ComPacket for the specified ExtendedComID with the Length, OutstandingData, and MinTransfer fields set per "All Response(s) returned – no further data" defined in Table 08.

State "Processing" – In this state, the TPer has begun processing the payload of an IF-SEND command.

While in this state, the TPer SHALL terminate any received or dequeued IF-SEND commands with "Synchronous Protocol Violation" as defined in [7].

While in this state, the TPer SHALL return a response ComPacket for any received or dequeued IF-RECV commands for the specified ExtendedComID with the Length, OutstandingData, and MinTransfer fields set per "Response(s) to come, no Response(s) available" defined in Table 08.

State "Awaiting IF-RECV" – The TPer has completely processed the TCG data payload and has the associated TCG response ready for retrieval by the host.

While in this state, if IF-RECV is received or dequeued with the ComID for this state machine and a transfer length less than the amount of response data staged for the ComID, the TPer SHALL return a response ComPacket for the specified ExtendedComID with the Length, OutstandingData, and MinTransfer fields set per "Response ready, insufficient transfer length request" defined in Table 08.

While in this state, the TPer SHALL terminate any received or dequeued IF-SEND command with a status of "Synchronous Protocol Violation" as defined in [7].

This specification defines the following protocol state transitions for each valid ComID:

"Power Off : Awaiting IF-SEND" - This transition occurs automatically when the TPer is powered on.

"Awaiting IF-SEND : Processing" - This transition occurs when an IF-SEND command with the ComID associated with this state machine is received or dequeued and successfully completes data transfer of the command payload.

"Awaiting IF-SEND : Power Off" - This transition occurs when the TPer is powered off.

"Processing : Awaiting IF-SEND" - This transition occurs when the TPer receives:

- an interface initiated TCG reset (see 4.5.1),
- a Protocol Stack Reset Command for the ComID of this state machine (see 4.5.2), or
- when the TPer detects an error in the IF-SEND payload that prevents the TPer from resolving an intended session for the IF-SEND command payload (see section 4.4.3.5) .

"Processing : Awaiting IF-RECV" - This transition occurs when the TPer has completely processed the contents of the IF-SEND command and has a complete response available for retrieval by the host. A separate response MAY be generated for each method in the IF-SEND.

"Processing : Power Off" - This transition occurs when the TPer is powered off.

"Awaiting IF-RECV : Awaiting IF-SEND" - This transition occurs when the TPer receives:

- an interface initiated TCG reset (see 4.5.1),
- a Protocol Stack Reset Command for the ComID of this state machine (see 4.5.2),
- an IF-RECV able to retrieve the entire response resulting from the IF-SEND, or
- an IF-RECV for the last of multiple responses resulting from the IF-SEND.

"Awaiting IF-RECV : Power Off" - This transition occurs when the TPer is powered off.

"Awaiting IF-RECV : Processing" - This transition occurs when the IF-SEND contained multiple method invocations, the TPer is preparing a separate response for each method, an IF-RECV able to retrieve the the current response was received and additional methods need to be processed.

4.4.2 Restrictions

This section defines the restrictions imposed on the exchange of IF-SEND and IF-RECV commands.

1. Any number of non IF-SEND/IF-RECV commands MAY be interleaved with IF-SEND/IF-RECV commands.
2. The normal communications state of an Associated ComID SHALL be to await receipt of an IF-SEND command for that ComID.
    a. While awaiting receipt of an IF-SEND interface command, any received IF-SEND command with a valid ComID SHALL be accepted.
    b. Once the entire command payload has been received, the TPer SHALL return an interface status to the host.
    c. Any IF-RECV command received for the Associated ComID awaiting receipt of an IF-SEND command SHALL return to the host a ComPacket with a Length field value of zero, an OutstandingData field value of zero, and a MinTransfer field value of zero. This signals to the host that there is no pending response data to retrieve.
3. After an IF-SEND command has been received, a command completion without error has been returned, and the payload has been decoded without an error, the TPer SHALL NOT accept another IF-SEND command for that ComID until the host has retrieved the entire response via IF-RECV(s).

a. Any subsequently received IF-SEND commands for the specified ComID SHALL be aborted at the interface level. The interface status for this action SHALL be specified in [7].

b. If the TPer has not sufficiently processed the command payload and prepared a response, any IF-RECV command for that ComID SHALL receive a ComPacket with a Length field value of zero (no payload), an OutstandingData field value of `0x01`, and a MinTransfer field value of zero.

c. If the TPer has sufficiently processed the command payload and prepared a response, an IF-RECV command that requests a transfer length less than the amount of response data the TPer has prepared SHALL reply with a ComPacket with a Length field value of zero (no payload) and OutstandingData value of total bytes currently available, and MinTransfer field value of zero or the minimum request required to transfer a packet.

d. In the case of TPers based on an SSC that permits multiple method invocations per IF-SEND command, the SSC MAY additionally require that each method response SHALL be retrieved separately (along with Control Tokens as determined by the TPer), via multiple IF-RECV commands. For these SSCs:

   i. If all responses have not been retrieved, and additional responses are available, the TPer SHALL respond to an IF-RECV command with OutstandingData value of total bytes currently available, and MinTransfer field value of zero or the minimum request required to transfer a packet.

   ii. If all responses have not been retrieved, and no additional responses are prepared but more are to come, the TPer SHALL respond to an IF-RECV command with OutstandingData field value of `0x01` and MinTransfer field value of Zero.

Table 08 IF-RECV Field Values

IF-RECV	Length Field Value	OutstandingData Field Value	MinTransfer Field Value
Response(s) to come, no Response(s) available	Zero	0x01*	Zero
Response ready, insufficient transfer length request	Zero	Total bytes currently available	Zero, or the minimum request required to transfer a packet
Response, additional Response(s) available	Data Length	Total bytes currently available	Zero, or the minimum request required to transfer a packet
Response, additional Response(s) to come, no Response(s) available	Data Length	0x01*	Zero
Response, all Response(s) returned – no further data	Data Length	Zero	Zero

IF-RECV	Length Field Value	OutstandingData Field Value	MinTransfer Field Value
All Response(s) returned – no further data	Zero	Zero	Zero

An OutstandingData field value of `0x01` denotes that the TPer is processing response(s). This provides insight to the host that there are responses still to come, but that are not ready yet. This is applicable to both the synchronous and asynchronous exchange of messages.

4.4.2.1 Error Handling

This section defines the manner in which violations of the restrictions on Interface Commands SHALL be handled by the TPer.

1. If a restriction violation occurs such that the TPer is unable to resolve a valid Session ID in an IF-SEND command, or if the restriction violation occurs due to violations of packet requirements, the TPer SHALL ignore the entire payload and SHALL immediately transition to the state of awaiting an IF-SEND command.
2. If a restriction violation occurs such that the TPer is able to resolve the Session ID, the TPer SHALL close that session and SHALL prepare for transmission the `CloseSession` method for retrieval by the host.
3. The device SHALL abort at the interface level any IF-SEND command whose transfer length is greater than the reported MaxComPacketSize for the corresponding ComID. The interface status for this action SHALL be specified in [7].
4. For SSCs that require that entire method responses be retrieved, if data generated in response to any single method in an IF-SEND command (together with required communications overhead) does not fit entirely within the TPer's response buffer, the device SHALL NOT return any part of that method response and SHALL instead return an empty response list with a status code of RESPONSE_OVERFLOW in the response status list. Additionally, the TPer SHALL continue processing methods and control tokens that had been sent in that command payload (if any).

4.4.2.2 Other Restrictions

There are two other restrictions necessitated by this restriction on the exchange of interface commands:

5. Methods SHALL NOT span ComPackets. In the case where an incomplete method is submitted, if the TPer can identify the associated session, then that session SHALL be aborted and a `CloseSession` MAY be prepared for delivery on Session 0/Session Manager Layer.
1. The synchronous exchange of interface commands SHALL only apply to IF-SEND/IF-RECV commands exchanged on Protocol ID 1.

4.4.3 Payload Encoding

4.4.3.1 Stream Encoding Modifications

The TPer SHALL support tokens listed in Table 09. If an unsupported token is encountered, the TPer SHALL treat this as a streaming protocol violation and return an error per the definition in section 4.4.3.5.

Table 09 Supported Tokens

Acronym	Meaning
	Tiny atom
	Short atom
	Medium atom
	Long atom
SL	Start List
EL	End List
SN	Start Name
EN	End Name
CALL	Call
EOD	End of Data
EOS	End of session
ST	Start transaction
ET	End of transaction

For supported atom tokens the TPer SHALL support token atoms with the B bit set to 0 or 1 and the S bit set to 0.

4.4.3.2 Short Atom Deviations

This section identifies deviations from the definition of Short Atoms in [2] that are required by this specification.

Short atoms consist of a one-byte header and between 0 and 15 bytes of data. A length of 0 SHALL only be permitted for non-continued bytes tokens.

Table 10 Short Atom Description

Header (1 byte)									Data		
Short Atom		byte/ integer	sign/ continued	length					(0...15 bytes)		
1	0	B	S	*n*	*n*	*n*	*n*		*d*	...	*d*

The encoding is as follows:

Table 11 Short Atom Encoding

Short Atom indicator	These two bits are set to 10b to indicate the atom is a short atom.
Byte/integer indicator	***Value*** ***Interpretation*** 0b The data bytes represent an integer value and the S bit indicates whether that value is signed. 1b The data bytes represent a byte sequence and the S bit indicates whether or not this value is continued into another atom.
Sign/continued indicator	***Value*** ***Interpretation*** 0b The interpretation of the data depends on the byte/integer indicator bit. B==0b The data is treated as unsigned integer data. B==1b The data is either the complete byte sequence, or the final segment of a continued byte sequence. 1b The interpretation of the data depends on the byte/integer indicator bit. B==0b The data is treated as signed integer data. B==1b The data is a non-final segment of a multi-byte continued value.
Length	These bits specify the length of the following data byte sequence. The permitted range is from 0 to 15, inclusive.

4.4.3.3 TCG Packets

Within a single IF-SEND/IF-RECV command, the TPer SHALL support a ComPacket containing one Packet, which contains one Subpacket. Host MAY discover TPer support of capabilites 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 (see 3.6.2).

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 (see 3.6.2).

The TPer MAY ignore TCG packet sequence numbering and not enforce any sequencing behavior. The discovery of TPer packet sequence numbering.support is outside the scope of this SSC.

4.4.3.4 Packetization Deviations

This section identifies deviations from the definition of Packets and Subpackets in [2] that are required by this specification.

4.4.3.4.1 Packet Modifications

Add a field of "**Reserved: uinteger_2**" between **SeqNumber** and **AckType** fields.

4.4.3.4.2 Subpacket Modifications

4.4.3.4.2.1 Subpacket Header

Move the **Reserved** field to come before the **Kind** field, and change the **Reserved** field to be of type **uinteger_6** in each of the defined Subpacket headers.

4.4.3.4.2.2 Credit Control Subpacket Modifications

The **Length** field in the Subpacket header SHALL contain a value of `0x00000004`.

In the Payload, the **Credit** field SHALL be changed to type **uinteger_4**.

4.4.3.4.2.3 Data Subpacket Modifications

In the payload, add a field after the **Data** field of "**Pad: bytes{(4 – (Length mod 4)) mod 4}**". The value of the Pad bytes SHALL be `0x00`.

Begin Informative Note

The receiver of a Subpacket can unambiguously know how many bytes of real data there are by examining the **Length** field in the Subpacket header. The receiver can also unambiguously know how many bytes of pad there are by calculating ((4 – (**Length** modulo 4)) modulo 4).

End Informative Note

4.4.3.5 Payload Error Response

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

- If the error is on 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 close the session and prepare a `CloseSession` method for retrieval by the host.

4.5 Storage Device Resets

4.5.1 Interface Resets

Interface resets that generate TCG reset events are defined in [7].

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;
7. No notification of these events SHALL be sent to the host.

4.5.2 Protocol Stack Reset Commands

An IF-SEND containing a Protocol Stack Reset Command SHALL be supported as defined in this section.

A new Request code is added to the HANDLE_COMID_REQUEST command as defined in [2]. The proposed Request code is STACK_RESET (`02h`) and its command block payload is defined as:

Bytes 0 to 3 :	Extended ComID value
Bytes 4 to 7 :	STACK_RESET (`00000002h`)
Bytes 8 to TRNSFLEN – 1:	Reserved (0s)

TRNSFLEN is defined as number of bytes transferred via the interface.

The device SHALL return an “Invalid Transfer Length parameter on IF-SEND” TPer Error as defined in [7] if less than 8 bytes or more than 512 bytes are sent to the device.

An Enterprise SSC compliant device MAY return:

an “Invalid ComID parameter on IF-SEND” Error, or

an “Other Invalid CDB parameter” Error (see [7]) if the ComID value in the IF-SEND for the HANDLE_COMID_REQUEST command represents a non Active ComID (see [2] for Active ComIDs).

Once received, the TPer SHALL reset the Security Protocol stack for the ComID value defined in bytes 0-3 of the command block payload. While resetting the stack, the Tper SHALL NOT process any command for that ComID received via an IF-SEND on Protocol ID `01h`. A Security Protocol stack reset results in:

1. All open sessions for that ComID SHALL be aborted;
2. All uncommitted transactions SHALL be aborted. `CloseSession` methods SHALL NOT be prepared by the TPer;
3. All pending session startup activities occurring on that ComID SHALL be aborted;
4. All TCG command and response buffers SHALL be invalidated for that ComID;
5. All related method processing occurring on that ComID SHALL be aborted;
6. The protocol stack SHALL reset to its initial state for that ComID only;
7. All communications properties (set via `Properties` method) and ComID associated properties for that ComID SHALL be reset to their default values;
8. No notification of these events SHALL be sent to the host.

The response SHALL be returned via the GET_COMID_RESPONSE (IF-RECV) command. The response payload is defined as:

Bytes 0 to 3:	Extended ComID
Bytes 4 to 7:	STACK_RESET (`00000002h`)
Bytes 8 to 9:	Reserved (`0000h`)
Bytes 10 to 11:	Available Data Length in bytes (`0004h`)
Bytes 12 to 15:	Success (`00000000h`) / Failure (`00000001h`)
Bytes 16 to TRNSFLEN - 1:	Reserved (0s)

Success (`00h`): the protocol stack has been reset for the specified ComID;
Failure (`01h`): the protocol stack has not been reset for the specified ComID;

A “Pending” payload is defined as:

Bytes 0 to 3:	Extended ComID
Bytes 4 to 7:	STACK_RESET (`00000002h`)
Bytes 8 to 9:	Reserved (0s)

Bytes 10 to 11:	Available Data Length in bytes (0000h)
Bytes 12 to TRNSFLEN - 1:	Reserved (0s)

A “No Response Available” payload is defined as:

Bytes 0 to 3:	Extended ComID
Bytes 4 to 7:	ZERO (00000000h)
Bytes 8 to 9:	Reserved (0s)
Bytes 10 to 11:	Available Data Length in bytes (0000h)
Bytes 12 to TRNSFLEN - 1:	Reserved (0s)

The response SHALL be cleared from the response buffer if one of the following conditions is true:

1. The host retrieves the entire response via the GET_COMID_RESPONSE command;
2. The device is hard-reset or power-cycled.
3. Another HANDLE_COMID_REQUEST is made for that ComID.

If the STACK_RESET is still processing and another HANDLE_COMID_REQUEST is received, the STACK_RESET SHALL complete but no response for that STACK_RESET command will be available.

Reserved bytes SHOULD be set to zero and SHALL be ignored by both host and device.

The device SHALL return “No Response Available” if:

1. No HANDLE_COMID_REQUEST command preceded the GET_COMID_RESPONSE command;
2. An error is detected in the HANDLE_COMID_REQUEST command payload.

The device SHALL return “Pending” if:

1. The host retrieves the command result via the GET_COMID_RESPONSE command while the stack reset is in progress for that specific ComID.

Begin Informative Note

The host is not required to retrieve the status via GET_COMID_RESPONSE, i.e. successful retrieval of the STACK_RESET response by the host does not have an effect on the execution of the command itself.

End Informative Note

5 Data Types

The TPer SHALL support TCG streaming protocol as defined in [2]. This section identifies deviations from the definition of the streaming protocol in [2] that are required by this specification. Specifically, this section defines the removal of type identifiers for parameter values from the messaging stream, clarifies the construction of method invocations and responses, and clarifies column type requirements.

This section defines the removal of type identifiers from the message stream (method parameters and results), while still enabling general identifiers for method parameter types through the use of basic interface types, and the use of types for columns. This includes the removal of method-associated types from the Type table.

5.1 Interface Types

Interface data types are introduced in [2]. This introduction is divided into several parts:

- Pseudo-Code – this section describes the formatting used in method signatures in [2].
- Messaging Data Types – this section introduces two data types used for messaging – Named values and List values.
- Method Parameter/Column Value Typing and Encoding – this section introduces the mechanism defined by [2] for inclusion of method parameter and result type identifiers in the message stream.

Because of the manner in which data is encoded and transferred across the interface, the actual types used in method parameter and result values can be described using a limited set of basic types: **Byte string values** and **N length integer values** (either signed or unsigned). All data is transferred across the interface as one of these two fundamental types (bytes or integers).

- **Byte-string values** are a sequence of n bytes that can be used to represent strings, blobs, bit vectors, etc.
- **N byte integer values** are whole numbers that can be either signed or unsigned.

Due to the nature of method parameters and results, there are two additional constructs defined for messaging that serve as grouping mechanisms for the fundamental types: **Named values** and **List values**.

- **Named values.** The name (a byte-value) followed by its value (any messaging type). Named values are used to send the optional parameters in method calls.
- **List values.** Zero or more values of some type, grouped into an ordered list. List values are used to encode method parameter lists and return results.

Method parameters and results are made up of byte and (signed or unsigned) integer values that can be grouped using Named values and List values.

5.2 Abstract Types

Abstract types are representations of grouped interface types, or interface types that have limits on their legal values. These representations are used primarily for documentation purposes, as part of the pseudo-code method signatures to simplify the description of those methods.

Abstract types do not affect the operation or regular encoding of a method, nor are they used as column types or represented in the Type table (though they MAY resemble some of these types in structure, name, or both). The primary goals of the abstract type constructs are to simplify the pseudo-code description of the methods themselves, and to provide insight into grouping using the List and Named value tokens introduced previously.

5.2.1 Abstract Types definitions

The following sections describe the pseudo-code parameters that each of these abstract types represent when they appear in a pseudo-code method signature.

5.2.1.1 access_control_list

An access_control_list is a list of uidrefs, specifically uidrefs to objects in the ACE table. The length of the list is implementation/SSC-specific.

Format:

```
[ uidref ... ]
```

5.2.1.2 boolean

This abstract type is similar to an enumeration column type, and has a valid range of the integer 0 to the integer 1, where 0 is used to represent "False" and 1 is used to represent "True".

Format:

```
uinteger
```

In the messaging stream, "False" will be represented as 0x00 and "True" will be represented as 0x01.

5.2.1.3 cell_block

This type represents a grouping of Named values that are used to identify a portion of a table. In messaging, this grouping is enclosed by List value delimiters, and each component is enclosed by Named value delimiters.

Because this is a group of Named values, its separate components are optional. However, there are default requirements if components are omitted. These requirements are as follows:

- Table – this Named value has the Name "Table" and a value that is a uid to a table.
    - If the value with Name "Table" is omitted, then the operation defaults to the table upon which the method was invoked.
    - Table SHALL be omitted if the method was invoked to operate on an object.
- startRow – this Named value has the Name "startRow". This Named value type can be assigned one of two values – either a uid of an object or a RowNumber that corresponds to the RowNumber value of an Array table row. Only one of these two values will appear in the messaging stream. The "typeOr" identifier and accompanying curly brackets ("{", "}") have no effect on the values as represented in the message.
    - If the value with Name "startRow" is omitted and the method is invoked to operate on a table, then the operation defaults to the first row of the table.
    - The value with Name "startRow" MAY be omitted if the method is invoked to operate on an object. If it is not omitted, it SHALL be the uid of the object on which the method is to operate, and SHALL be the same as the value assigned to endRow.
    - If both the value with Name "startRow" and the value with Name "endRow" are included in the type parameterization, then the value with Name "startRow" SHALL have the same type (uid or uinteger) as the value with Name "endRow".
- endRow – this Named value has the Name "endRow". This Named value type can be assigned one of two values – either a uid of an object or a RowNumber that corresponds to the RowNumber value of an Array table row. Only one of these two values will appear in the messaging stream. The "typeOr" identifier and accompanying curly brackets ("{", "}") have no effect on the values as represented in the message.

- If the value with Name "endRow" is omitted and the method is invoked to operate on a table, then the operation defaults to the last row of the table.
- The value with Name "endRow" SHALL be omitted if the method is invoked to operate on an object. If it is not omitted, it SHALL be the uid of the object on which the method is to operate, and SHALL be the same as the value assigned to startRow.
- If both the value with Name "startRow" and the value with Name "endRow" are included in the type parameterization, then the value with Name "endRow" SHALL have the same type (uid or uinteger) as the value with Name "startRow".

- startColumn – this Named value has the Name "startColumn". This Named value type has a max bytes value that is represented by here using the name abstract type.
  - If the value with Name "startColumn" is omitted, then the operation defaults to the first column of the table or object.
- endColumn – this Named value has the Name "endColumn". This Named value type has a max bytes value that is represented by here using the name abstract type.
  - if the value with Name "endColumn" is omitted, then the operation defaults to the last column of the table or object.

Format:

```
[ Table = uid, startRow = typeOr { UID : uid, Row : RowNumber }, endRow = typeOr
{ UID : uid, Row : RowNumber }, startColumn = name, endColumn = name ]
```

5.2.1.4 date

This type represents a grouping of Named values that are used to identify time values, and is similar to the column type of the same name. In messaging, this grouping is enclosed by List value delimiters, and each component is enclosed by Named value delimiters.

Because this is a group of Named values, its separate components are optional. Components that are omitted are considered to have a value of 0.

The components are as follows:

- Year – this Named value has the Name "Year" and a value that is implicitly defined as being of uinteger of size 2. This Named value abstract type represents the year in a timestamp. Valid values are unsigned integers ranging from 1970 to 9999
- Month – this Named value has the Name "Month" and a value that is implicitly defined as being of uinteger of size 2. This Named value abstract type represents the month in a timestamp. Valid values are unsigned integers ranging from 1 to 12, which correspond to the months of the year as follows:
  - January = 1 (0x01)
  - February = 2 (0x02)
  - March = 3 (0x03)
  - April = 4 (0x04)
  - May = 5 (0x05)
  - June = 6 (0x06)
  - July = 7 (0x06)
  - August = 8 (0x08)

- September = 9 (0x09)
- October = 10 (0x0A)
- November = 11 (0x0B)
- December = 12 (0x0C)

- Day – this Named value has the Name "Day" and a value that is implicitly defined as being of uinteger of size 1. This Named value abstract type represents the day of the month in a timestamp. Valid values are unsigned integers ranging from 1 to 31.

Format:

```
[ Year = uinteger, Month = uinteger, Day = uinteger ]
```

5.2.1.5 name

This type is a representation of the max bytes type, and in most methods in which it is used it is assigned to parameters that are associated with a table's `Name` column or `CommonName` column. As such, it has an implicit size restriction of 32 bytes.

Format:

```
bytes
```

5.2.1.6 ref

The ref abstract type represents a reference to a table row that is expressed using a uinteger type with a size of 8, and corresponds to a row's `RowNumber` column value.

In the pseudo-code method signatures, the ref abstract type is often followed by curly brackets ("{", "}") that are used to define the limitation of a valid value for that ref. These valid values are typically represented as a reference to a specific table, which indicates that to ultimately be considered valid, the ref must be to a row in that table.

Limitations expressed with curly brackets have no effect on the appearance of the associated ref value as it appears in the message stream. Because this abstract type describes the inclusion of a RowNumber, it represents a uinteger value that has an implicit size restriction of 8 bytes in the uinteger value.

Format:

```
uinteger
```

5.2.1.7 row_address

This abstract type is used to describe a parameter that can be either a ref or a uidref. It is similar to the alternative column type. For additional information on the component types (ref and uidref), see their respective entries in this section.

Only one of these two values will appear in the messaging stream. The "typeOr" identifier and accompanying curly brackets ("{", "}") have no effect on the values as represented in the message.

Format

```
typeOr { RowAddress : ref, UIDAddress : uidref }
```

In the message stream itself, the value will one of the following:

- `ref`
- `uidref`

5.2.1.8 row_data

This type represents a list of lists of Named values. Each interior list represents a row, so there are multiple interior lists (a list of lists). The Named values represent column names and the values to be associated with them. The number of interior lists (i.e. the number of rows that MAY be represented by this type "at one time") MAY be limited by SSC or implementation.

Format:

```
[ [ ColumnName = Value ... ] ... ]
```

5.2.1.9 uidref

The uidref abstract type represents a uid of an object, table, or table row that is expressed using a bytes type with a size of 8, and corresponds to an object, table, or table row's UID column value.

In the pseudo-code method signatures, the uidref abstract type is often followed by curly brackets ("{", "}") that are used to define the limitation of a valid value for that uidref. These valid values are typically represented as requiring an object of a specific type. Limitations expressed with curly brackets have no effect on the appearance of the associated uid value as it appears in the message stream.

Because this abstract type describes the inclusion of a uid, it represents a bytes value that has an implicit size restriction, and that value SHALL always be 8 bytes long.

Format:

```
bytes
```

6 Method Status Code Deviations

This section defines deviations from [2] pertaining to Status Codes as required by this specification.

Table 12 Status Codes

Name	Value
AUTHORITY_LOCKED_OUT	0x12

6.1 AUTHORITY_LOCKED_OUT

This status MAY be returned as the status code of the `SyncSession` method or in response to the `Authenticate` method under one of the following conditions:

1) If an authority with the `Operation` column value of Password is being authenticated and its associated `C_PIN` object has a `Tries` column value equal to its `TryLimit` column value, and the `TryLimit` column is not set to 0; or
2) If the `Uses` column of the authority being authenticated has reached the value of its `Limit` column, and the `Uses` column is not set to 0.

7 Method Signatures

Method signatures are presented in pseudo-code, which is used to describe types, method parameters, and snippets of code without having to use the byte encodings directly.

Methods are made up of two kinds of parameters: required and optional.

- Required parameters must come in the order given in the method signature, and must precede optional parameters. In the pseudo-code signature, required parameters are given expositional names for ease of reference. The right-hand portion of the parameter is the interface or abstract type that SHALL be used with that parameter.

  Required parameters are formatted as follows:

  - `Expositional-Name : Parameter-type`

- Optional parameters are required to come in order, and each SHALL appear only once in a method invocation or the method SHALL fail and return a non-Success status. In the pseudo-code signature, optional parameters are given in the form of Named values except the right-hand portion of the parameter is the interface or abstract type that SHALL be used with that parameter. The Name (left-hand portion in the pseudo-code) SHALL be the name of the Named value type for that parameter when the method is invoked.

  Optional parameters are formatted as follows:

  - `Parameter-Name = Parameter-type`

The result portion of a method's signature are formatted similarly to the parameters, using the same conventions for results that are required to be returned for successful method invocations ("required results"), and results that MAY be returned only in certain situations ("optional results"). The result list of a failed method invocation should be empty.

Any appearance of "=" in a method's parameter list or result list (including in abstract type definitions) indicates the required use of an interface Named value, where the required Name is to the left of the "=" and the required value is to the right of the "=".

Parameters typically have implicit size restrictions based on the table column that the particular parameter is modifying or to which it is referring.

Separating brackets ("[", "]") in method signatures are used to mark places in the stream where List values are used to encapsulate values. Commas (",") in the pseudo-code method signatures are used to separate items in a list. Ellipses in pseudo-code method signatures are used to indicate multiples of the immediately preceding type appears within the list (i.e. within the closest set of enclosing brackets).

In the pseudo-code, curly braces ("{", "}") are used to signify additional information regarding the type with which they are associated, but are not required to be checked as part of method parsing and do not affect the content or composition of the messaging stream.

7.1 Session Manager

7.1.1 StartSession/SyncSession

```
SMUID.StartSession [
        HostSessionID : uinteger,
        SPID : uidref {SPObjectUID},
        Write : boolean,
```

```
            HostChallenge = bytes,
            HostExchangeAuthority = uidref {AuthorityObjectUID},
            HostExchangeCert = bytes,
            HostSigningAuthority = uidref {AuthorityObjectUID},
            HostSigningCert = bytes,
            SessionTimeout = uinteger,
            TransTimeout = uinteger,
            InitialCredit = uinteger,
            SignedHash = bytes ]
      =>
      SMUID.SyncSession [
            HostSessionID : uinteger,
            SPSessionID : uinteger,
            SPChallenge = bytes,
            SPExchangeCert = bytes,
            SPSigningCert = bytes,
            TransTimeout = uinteger,
            InitialCredit = uinteger,
            SignedHash = bytes ]
```

7.1.2 CloseSession

```
      SMUID.CloseSession [
            RemoteSessionNumber : uinteger,
            LocalSessionNumber : uinteger ]
```

7.2 Base Template

7.2.1 Get

```
      TableUID.Get [
      ObjectUID.Get [
            Cellblock : cell_block ]
      =>
      [ Result : typeOr { Bytes : Bytes, RowValues : list [ list [ ColumnName = Value
... ] ... ] } ]
```

7.2.2 Set

```
      TableUID.Set [
      ObjectUID.Set [
            Where : cell_block,
            Values : typeOr { Bytes : bytes, RowValues : list [ list [ ColumnName =
Value ... ] ... ] } ]
```

```
      =>
      [ Result : boolean ]
```

7.2.3 Next

```
      TableUID.Next [
            Where = row_address,
            Count = uinteger ]
      =>
      [ Result : TypeOr { ArrayTable : list [ [ ref, uidref ] ... ], ObjectTable : list
[ uidref ... ] } ]
```

7.2.4 Authenticate

```
      SPUID.Authenticate [
            Authority : uidref { AuthorityObjectUID },
            Challenge = bytes ]
      =>
      [ Result : typeOr { Success : boolean, Response : bytes } ]
```

7.2.5 GetACL

```
      MethodTableUID.GetACL [
            InvokingID : uidref { SP/table/object },
            MethoID : uidref { MethodID } ]
      =>
      [ Result : access_control_list ]
```

7.3 Crypto Template

7.3.1 Random

```
      SPUID.Random[
            Count : uinteger,
            BufferOut = cell_block ]
      =>
      [ Result : bytes ]
```

8 Column Types in Messaging

Certain column types used in messaging as method parameters (particularly in the Set method) utilize the interface grouping mechanisms (Named and List values) to provide clarity regarding the scope of the transmitted values.

- Simple types – values of this type require no special handling in the messaging stream.
- Enumeration types – values of this type require no special handling in the messaging stream.
- List type – the "List" column type is handled in the same way a parameter list is handled, by using the interface List value grouping tokens (F0 and F1 tokens, which represent "[" and "]" respectively) to enclose the values in the list.
- Restricted Reference types – values of this type require no special handling in the messaging stream.
- General Reference types – values of this type require no special handling in the messaging stream.
- Set value types – the "Set" column type is handled in the same way that the List type is handled, by using the interface List value grouping tokens (F0 and F1 tokens, which represent "[" and "]" respectively) to enclose the values in the Set.

9 Session Manager

9.1 Session Timeouts

During session startup, if the host specifies a timeout outside of the supported TPer timeout interval, the TPer rejects the session startup command and returns a failed `SyncSession` method call with TCG status INVALID_PARAMETER.

9.2 Session Manager Method Requirements

TPer support for the Session Manager methods in Table 13.

Table 13 Session Manager Methods

Method Name	Method Type
Properties	Session Manager
StartSession	Session Manager
SyncSession	Session Manager
CloseSession	Session Manager

9.2.1 Session Manager Deviations

This section defines deviations from [2] pertaining to Session Manager methods as required by this specification.

All Session Manager Layer Methods SHALL be transmitted in packets where Packet.SessionNumber = `0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00`.

Once a session has started (the session startup protocol has completed successfully), data MAY be transmitted for that newly started session. The Packet.SessionNumber for that session SHALL be the concatenation of the TSN and HSN, as described in [2], where HSN is initially transmitted in the `StartSession` method and TSN is initially transmitted in the `SyncSession` method.

9.2.2 Session Manager Methods

9.2.2.1 Properties

The TPer SHALL implement the `Properties` method with the constraints stated in this subsection.

When a `Properties` method is received, the TPer SHALL return the following parameters:

MaxPacketSize	= min 1024 bytes - 20 bytes (ComPacket Hdr)
MaxComPacketSize	= min 1024 bytes
MaxResponseComPacketSize	= min 1024 bytes
MaxSessions	= min 1
MaxIndTokenSize	= min 256 bytes
MaxAuthentications	= min 2
MaxTransactionLimit	= min 1

These values represent the minimum communications capabilities that the TPer supports for messages it receives. The values listed above for MaxPacketSize, MaxComPacketSIze, and MaxIndTokenSize also apply to communications from the TPer to the Host.

The TPer SHALL ignore any parameters not supported by the TPer when the host tries to set its value and not include it nor its value in the return data (behavior specified in the [2]).

The TPer SHALL return all property name/value pairs for capabilities that it supports. For capabilities not supported by the TPer (e.g, Read-Only sessions), the associated property name/value pair (in this case, MaxReadSessions) SHALL be omitted from the TPer's response.

9.2.2.1.1 Properties Method Deviations

This section defines deviations from [2] pertaining to the operation of the `Properties` method as required by this specification.

The `Properties` method pertains to the exchange of session-related metadata and settings between the host and the TPer prior to session start-up. The purpose of the `Properties` method is to permit the host and the TPer to exchange the information required for session startup and maintenance, without the need to first start a session.

```
SMUID.Properties[ HostProperties = list [ name = value ... ] ]
=>
SMUID.Properties[ Properties : list [ name = value ... ], HostProperties = list [ name =
value ... ] ]
```

This Session Manager layer method is used by the host to provide its communication properties to the TPer, and to retrieve the communication properties of the TPer.

A list of name/value pairs MAY be provided as the optional HostProperties argument when invoking the `Properties` method.

If the method is successfully invoked the response is a list of property names and values from the TPer.

The TPer SHALL return all property name/value pairs for capabilities that it supports. For capabilities not supported by the TPer (for instance, Read-Only sessions), the associated property name/value pair (in this case, MaxReadSessions) SHALL be omitted from the TPer's response.

The TPer MAY also respond with additional name/value pairs other than those specified in this document.

The order of the name/value pairs returned by the TPer is not specified.

For the name/value pairs returned by the TPer, the TPer SHALL return values for the associated names as described in Table 14 or in the associated SSC (the values in the SSC have precedence). The values returned SHALL apply to all sessions started with the currently associated ComID.

If the method is invoked with the optional HostProperties parameter, the list of name/value pairs that the TPer SHALL recognize is:

- **MaxSubpackets**
- **MaxPacketSize**
- **MaxPackets**
- **MaxComPacketSize**
- **MaxResponseComPacketSize –** for Enterprise SSC-compliant devices, responding to this host property is Not Required (N)
- **MaxIndTokenSize**

- **MaxAggTokenSize –** for Enterprise SSC-compliant devices, responding to this host property is Not Required (N)

These parameters are used to describe the communications capabilities that the host possesses, and apply to any sessions started using the ComID associated with this `Properties` method invocation once the TPer has processed the request

These values MAY be submitted in any order by the host. Not all values are required to be submitted. Subsequent submission of these values (in a subsequent invocation of the `Properties` method) SHALL supersede values submitted to previous invocations of the `Properties` method for that ComID. Submitted values, if applicable, SHALL only apply to sessions started after the submission of those values, and not to sessions that are already open on that ComID.

The TPer MAY use these host properties when it is constructing responses to be transmitted to the host. The host MAY omit properties as necessary, depending on the host's communications capabilities. If the host omits a property, or specifies a value for a property that does not meet the minimum requirement as defined in Table 14, then the TPer SHALL use the minimum value defined in 9.2.2.1in place of the value supplied by the host.

If the host includes the HostProperties parameter in the `Properties` method invocation, then the TPer's response SHALL include all communication property value settings, including those it will use during any subsequently started sessions (both for its communications and the host's). These values reflect the cumulative modifications of all processed `Properties` methods for the associated ComID.

If a host includes property parameters to the `Properties` method invocation that the TPer does not recognize or comprehend, the TPer SHALL ignore those parameters, and SHALL NOT return them in its response.

Because of the session-less nature of the Session Manager protocol layer, and the possible different ordering of responses to Session Manager layer methods, the response to this method is formatted as a Properties method invocation so as to be identifiable as the response to the `Properties` method.

Begin Informative Note

It is the host's responsibility to insure that `Properties` method invocations have processed prior to invocation of any session startup methods that rely on those invocations. Values for HostProperties at session startup rely on the `Properties` method invocations that have been processed by the TPer.

End Informative Note

Table 14 Properties Method Response

Property	Type	Description
MaxMethods	uinteger	Identifies the maximum number of methods the TPer SHALL accept in a single subpacket. A value of 0 indicates no limit.
MaxSubpackets	uinteger	Identifies the maximum number of subpackets that the communicator SHALL accept in a single Packet. A value of 0 indicates no limit (both a TPer Property and a Host Property).
MaxPacketSize	uinteger	The maximum size of a packet (including both data and header), in bytes, that the communicator is able to receive. This value SHALL be at least 512-ComPacketHeader overhead. A value of 0 indicates no limit (both a TPer Property and a Host Property).

Property	Type	Description
MaxPackets	uinteger	Identifies the maximum number of packets that the communicator SHALL accept in a single ComPacket. A value of 0 indicates no limit (both a TPer Property and a Host Property).
MaxComPacketSize	uinteger	The maximum size of an IF Command payload (includes both the ComPacket header and payload) that the communicator is able to receive. This value SHALL be at least 512. A value of 0 indicates no limit (both a TPer Property and a Host Property).
MaxResponseComPacketSize	uinteger	The maximum length of an IF Command payload that the communicator is able to generate. A value of 0 indicates no limit (both a TPer Property and a Host Property).
MaxSessions	uinteger	The maximum number of simultaneous sessions supported by the TPer. A value of 0 indicates no limit.
MaxReadSessions	uinteger	The maximum number of simultaneous Read-Only sessions to any one SP supported by the TPer. A value of 0 indicates no limit.
MaxIndTokenSize	uinteger	The maximum size of a token (in bytes) in a single subpacket that the communicator is able to accept. Token size refers to both the token header and data. This value SHALL be at least 256. A value of 0 indicates no limit (both a TPer Property and a Host Property).
MaxAggTokenSize	uinteger	The maximum aggregate size of a continued token after all individual parts of that token are combined that the communicator is able to accept. Token size refers to both the token header and data. This value SHALL be at least 256. A value of 0 indicates no limit (both a TPer Property and a Host Property).
MaxAuthentications	uinteger	The maximum number of simultaneously authenticated individual authorities per session that the TPer is able to support. A value of 0 indicates no limit.
MaxTransactionLimit	uinteger	The maximum number of concurrently open transactions that the TPer is able to support in a single session. A value of 0 indicates no limit.
DefSessionTimeout	uinteger	The session timeout length (in milliseconds) used by the TPer by default. A value of 0 indicates no limit.
MaxSessionTimeout	uinteger	The longest supported session timeout length (in milliseconds) supported by the TPer. A value of 0 indicates no limit.

Property	Type	Description
MinSessionTimeout	uinteger	The shortest supported session timeout length (in milliseconds) supported by the TPer. A value of 0 indicates no limit.
DefTransTimeout	uinteger	The transmission timeout length (in milliseconds) used by the TPer by default. A value of 0 indicates no limit.
MaxTransTimeout	uinteger	The longest transmission timeout length (in milliseconds) permitted by the TPer. A value of 0 indicates no limit.
MinTransTimeout	uinteger	The shortest transmission timeout length (in milliseconds) permitted by the TPer. A value of 0 indicates no limit.
MaxComIDTime	uinteger	The timeout length (in milliseconds) used by the TPer after it has assigned a ComID. A session using the associated ComID SHALL be started within this interval or the ComID SHALL transition from Issued to Inactive. A value of 0 indicates no limit.
MaxComIDCMD	uinteger	SSC-dependent limit on the number of interface commands that MAY be issued using a specific ComID, including both IF-SEND and IF-RECV commands. A value of 0 indicates no limit.

9.2.2.2 StartSession

The TPer SHALL implement the `StartSession` method with the constraints stated in this subsection.

TPer support of the following parameters is mandatory:

- HostSessionID
- SPID
- Write (support of Write = True mandatory)

Attempts to use unsupported parameters SHALL result in a `SyncSession` response with TCG status INVALID_PARAMETER.

9.2.2.3 SyncSession

The TPer SHALL implement the `SyncSession` method with the constraints stated in this subsection.

Device support of the following parameters is mandatory:

- HostSessionID
- SPSessionID

9.2.2.4 CloseSession

The `CloseSession` method on the session manager SHALL only be invoked by the TPer in response to an erroneous IF-SEND from the host.

10 Templates

10.1 Definitions

For the purpose of this section, the following definitions SHALL apply:

- The TPer returns an INVALID_COMMAND TCG status when the host attempts to invoke a method designated “Excluded”.
- The TPer returns a NOT_AUTHORIZED TCG status when the host attempts to invoke a method on a table/object not permitted by the access control definitions.

10.2 Supported Templates

The TPer SHALL support the following modified templates:

- Base
- Admin
- Locking
- Crypto

10.2.1 TPer Template Requirements

The template requirements in Table 15 are mandatory.

Table 15 TPer Templates

Template Name	SSC Reference Section	Minimum Number Instantiable	Maximum Number Instantiable
Admin	Section 10.4	1	1
Base	Section 10.3	2	VU
Locking	Section 10.5	1	1
Crypto	Section 7.6	2	VU

10.3 Base Template

This subsection defines the modified Base Template as applicable for this specification.

10.3.1 Base Template Table Requirements

Support for the Base Template table access requirements in Table 16 is mandatory.

Table 16 Base Template Tables

Table Name	Table Type
Authority	Object
C_PIN	Object

10.3.1.1 Base Template Table Deviations

This section defines deviations from [2] regarding table naming as required by this SSC.

In [2], the `Method` table is a table that contains access control associations between methods and entities (objects/tables/SP). In this document, the `Method` table is referred to as the `AccessControl` table.

10.3.2 Base Template Method Requirements

Support for the Base Template method requirements in Table 17 is mandatory.

Table 17 Base Template Methods

Method Name	Method Type
Get	Object
Set	Object
Next	Table
Authenticate	SP
GetACL	Table

10.3.2.1 Method Deviations

This section defines deviations from [2] regarding method parameter encoding as required by this SSC.

[2] indicates:

> "Optional parameters are not required to be in order, and are not required to be included in a method invocation."

For all methods in a storage device that implements this SSC, if any optional parameters of a method are supplied to an invocation of that method, the supplied optional parameters SHALL be provided in the order specified [2] or this SSC for that method. If any optional parameter is supplied out of order, the method invocation SHALL fail and return INVALID_PARAMETER.

10.3.3 Base Template Method Details

10.3.3.1 Get

The UID for the `Get` method is: = `00 00 00 06 00 00 00 06`

The TPer SHALL implement the `Get` method with the constraints stated in section 5 and modified according to the below definition:

```
TableUID.Get [
ObjectUID.Get [
      Cellblock : cell_block,
      ParamCheck = boolean ]
=>
[ Result : typeOr { Bytes : Bytes, RowValues : list [ list [ ColumnName = Value ... ]
... ] },
```

```
ParamCheck = uinteger_2 ]
```

Method behavior is modified as follows:

The host MAY use the ParamCheck parameter to request a check value for a PIN credential value in the `Get` method result.

If the ParamCheck parameter value is True, the TPer SHALL perform a check value calculation as defined in section 10.3.3.3 on the result values portion of the method return result. The check value is returned as the ParamCheck portion of the method result.

If the ParamCheck parameter is omitted or if its value is False, the ParamCheck portion of the method result SHALL be omitted and only the contents of the requested Cellblock returned.

If the get_result contains no data, the TPer SHALL NOT return the ParamCheck Name-Value pair.

For a `Get` method example of ParamCheck calculations, see 13.2.

10.3.3.2 Set

The UID for the `Set` method is: = `00 00 00 06 00 00 00 07`

The TPer SHALL implement the `Set` method with the constraints stated in section 5 and modified according to the below definition:

```
Table.Set [
Object.Set [
      Where : cell_block,
      Values : typeOr { Bytes : bytes, RowValues : list [ list [ ColumnName = Value ...
] ... ] },
      ParamCheck = uinteger_2 ]
=>
[Result : boolean ]
```

Method behavior is modified as follows:

The host SHALL be required to provide at least one value to set in the "Values" parameter. In the absence of any Values, the TPer SHALL return INVALID_PARAMETER.

For Byte and Array tables, the TPer MAY ignore EndRow in the Where parameter.

For Array tables, Object tables, and Objects, the TPer MAY ignore StartColumn and EndColumn in the Where parameter.

The host MAY use the ParamCheck parameter to provide a check value for a PIN credential value.

If the ParamCheck parameter is supplied, the TPer SHALL first perform a calculation as defined in section 10.3.3.3. The TPer SHALL compare its computed value with that supplied in the ParamCheck parameter. If the TPer computed value matches the supplied ParamCheck value, then the device SHALL continue execution of the `Set` method. If the TPerTPer's computed value does not match the supplied ParamCheck value, then the method SHALL fail with TCG status INVALID_PARAMETER.

The TPer SHALL NOT calculate nor validate a check when the ParamCheck parameter value is omitted.

For a `Set` method example of ParamCeck calculations, see 13.1.

.

10.3.3.3 Check Value Algorithm

The TPer SHALL implement the ParamCheck Longitudinal Redundancy Check (LRC) for `Get` and `Set` method calls on a PIN value. The LRC word size is 16-bits and the seed (initial value) is `0x5056`.

If the parameter is submitted to a `Get` or `Set` method invocation that operates on anything other than the `PIN` column of a `C_PIN` object, such as when the invoking ID of the method is not a `C_PIN` object or a column other than the `PIN` column is operated on, the method SHALL fail and return a response of INVALID_PARAMETER.

Only the PIN parameter value is used for the LRC calculation. Tokens, names and ParamCheck parameter are not input to the LRC calculation.

Words are input to the LRC calculation in stream order, i.e. in Big Endian order.

If a PIN parameter is not an even number of bytes in length, a pad byte of `0x00` is prepended as the MSB.

The calculation algorithm is:

- Initialize the LRC calculation (initial value `0x5056`)
- For value in list of values
  - Convert value into its TCG byte representation using minimal width stream encoding
  - Discard any bytes associated with TCG token headers
  - If sizeof(value) is not a multiple of 2
    - Prepend one byte equal to `0x00` to the MSB of the value
  - Feed value into the LRC calculation
- Get final LRC result

In pseudo code, the above MAY be expressed as:

```
lrc = 0x5056  // Initialize LRC
for value in values   // ‘values’ represents all values from any name/value
                      // pairs in the Values parameter of the Set method
                      // or in the get_result of the Get method.
   bytes = value.tcg_encode()  // Convert value to its TCG encoding
   bytes = bytes.strip_tcg_header() // Remove TCG token headers
   if len(bytes) modulo 2 != 0
      bytes = bytes.prepend(0x00)
   // What follows is the heart of the LRC calculation…
   for word in bytes  // ‘word’ is 2 bytes
      lrc = lrc xor word
return lrc
```

10.3.3.4 Authenticate Deviations

This section identifies deviations from [2] regarding the operation of the `Authenticate` method as required by this specification.

10.3.3.4.1 Parameters

[2] indicates that, when an authority that is being authenticated by the host through the use of the Authenticate method requires challenge-response, the host must invoke the `Authenticate` method twice. The first `Authenticate` SHALL be empty, and the result should be a challenge from the TPer.

For SDs implementing this SSC, The first `Authenticate` method in the challenge/response pair SHALL be invoked with an Authority object UID as the Authority parameter, so that the TPer is able to determine the length of the challenge to be returned as the result of that method invocation.

The second `Authenticate` method invocation SHALL also be invoked with an Authority object UID as the Authority parameter, which SHALL be the same as that used in the first invocation. In addition, as specified, the second `Authenticate` method invocation SHALL contain the response to the TPer's challenge as its Challenge parameter.

10.3.3.4.2 Transactional Behavior

Successful invocations of the `Authenticate` method occur outside of transactional control, such that even in the event that a transaction in which a successful `Authenticate` method occurs is aborted, the authority authenticated by that method invocation continues to be authenticated.

If a successful `Authenticate` method invocation is made at any time within a session (either inside or outside of a transaction), the authority is considered authenticated for the rest of the session and any subsequent method invocations that depend on that authentication will be authorized. This applies even to a successful `Authenticate` method invocation that occurs in a transaction that is subsequently aborted.

10.3.3.4.3 Authenticate Method Failures

`Authenticate` returns different status codes and return results dependant on the success or failure of the method.

The nature of the `Authenticate` method prescribes two states:

a. Awaiting Challenge

b. Awaiting Challenge Response

Behavior of the `Authenticate` method when the SP is in the Awaiting Challenge state:

1. The method returns INVALID_PARAMETER with an empty result list if the following conditions apply, and remains in the Awaiting Challenge state:
    a. There is no authority supplied to the method, or an invalid authority (as in, an authority that does not exist in the `Authority` table) is supplied to the method, or
    b. A class authority is supplied to the method, or
    c. An incorrect optional parameter identifier or extra parameters are supplied, or a proof is supplied to a non-Password/non-Anybody authority.
2. If the conditions defined in 6.1 are met, the method SHALL either return AUTHORITY_LOCKED_OUT with an empty result list and remain in the Awaiting Challenge state, or follow the result and status behavior defined in 5 below.
3. If the `Authenticate` invocation does not violate any of the conditions in 1 and 2, above, and if the following conditions are met, then the method returns SUCCESS with a result of True (authentication succeeded), and remains in the Awaiting Challenge state:
    a. The authority invoked in the method is a valid individual authority object in the `Authority` table.

b. All of the authority's attributes are appropriate for the existing session and authentication attempt – the secure messaging properties are appropriate, the authority is enabled, has an `Operation` column value of Password, etc.

c. The authority references a valid `C_PIN` credential or the authority is the Anybody authority.

d. For authorities other than the Anybody authority, the correct password was submitted to the `Authenticate` method invocation.

4. If the `Authenticate` invocation does not violate any of the conditions in 1 and 2 above, and if the following conditions are met, then the method returns SUCCESS with a result of a 32-byte challenge, and transitions to the Awaiting Challenge Response state:

   a. The authority invoked in the method is a valid individual authority object in the `Authority` table.

   b. All of the authority's attributes are appropriate for the existing session and authentication attempt – the secure messaging properties are appropriate, the authority is enabled, references a valid credential, etc.

5. If the `Authenticate` invocation does not violate any of the conditions in 1 and 2 above, and if the following conditions are met, then the method returns SUCCESS with a result of False (authentication failed), and remains in the Awaiting Challenge state:

   a. The authority is a valid authority but one or more of its attributes are not appropriate or valid – the secure messaging requirements have not been met, the authority is disabled, the authority references an invalid credential, the authority has an `Operation` column value of Exchange or TPerSign, etc.

   b. The authority is a valid authority but an incorrect password is submitted to the `Authenticate` method invocation.

Behavior of the `Authenticate` method when the SP is in the Awaiting Challenge Response state:

1. The method returns INVALID_PARAMETER with an empty result list if the following conditions apply, and transitions to the Awaiting Challenge state:

   a. There is no authority supplied to the method, or an invalid authority (as in, an authority that does not exist in the `Authority` table) is supplied to the method

   b. An incorrect optional parameter identifier or extra parameters are supplied.

2. If the `Authenticate` invocation does not violate any of the conditions in 1 above, and if the following conditions are met, then the method returns SUCCESS with a result of True (authentication succeeded), and transitions to the Awaiting Challenge state:

   a. The authority invoked in the method is a valid individual authority object in the `Authority` table.

   b. The authority invoked in the method is the same authority invoked in the initial `Authenticate` invocation.

   c. All of the authority's attributes are appropriate for the existing session and authentication attempt – the secure messaging properties are appropriate, the authority is enabled, references a valid credential, etc.

   d. The correct response was submitted to the `Authenticate` method invocation.

3. If the `Authenticate` invocation does not violate any of the conditions in 1 above, and if the following conditions are met, then the method returns SUCCESS with a result of False (authentication failed), and transitions to the Awaiting Challenge state:

a. The authority is a class authority.

b. The authority invoked in the method is not the same authority invoked in the initial `Authenticate` invocation.

c. The authority is a valid authority but one or more of its attributes are not appropriate or valid – the secure messaging requirements have not been met, the authority is disabled, the authority references an invalid credential, the authority has exceed its uses, etc.

d. The authority is a valid authority but an incorrect challenge response is submitted to the `Authenticate` method invocation.

10.3.3.5 Next

This section identifies deviations from the [2] regarding the operation of the `Next` method as required by this specification.

When successfully invoked on an array table, the `Next` method returns zero or more row number/uidref pairs currently in use in the table following the specified **Where** row, iterating sequentially (by `RowNumber` column value) through the table rows. If there are fewer than **Count** rows defined after the indicated starting row, only the defined row numbers are returned.

The `Next` method MAY be used to discover an ordering of rows in an object table. Since the ordering of object tables is unspecified, the ordering that is discovered by successful invocation(s) of this method on an object table will be some undefined ordering, the “current” ordering.

When successfully invoked on an object table, the `Next` method returns a list of zero or more uidrefs “following” the specified **Where** row in the current ordering. If a value for the **Where** parameter is not specified in the method invocation, the first element, if any, of the list of uidrefs, will denote the “beginning” of the ordering, i.e. the row which has no predecessor in the current ordering.

The implementation is required to discover a consistent ordering of all rows of an object table only if the object table is not modified between calls to `Next`. Actions which cause modifications to the object table that would result in a new ordering SHALL be specified in each SSC, and SHALL include at least method calls adding or deleting rows if those are permitted by the SSC.

- If both the Where parameter and the Count parameter are omitted, the scope of the `Next` method's return value is the entire table.
- If the Where parameter is included in the invocation and the Count parameter is omitted, the scope of the `Next` method's return value begins at the starting point in the table's ordering indicated by the row following that identified by the Where parameter, and ends at the end of the table's row ordering.

10.4 Admin Template

This subsection defines the modified Admin Template as applicable for this specification.

10.4.1 Admin Template Table Requirements

There are no Admin Template table requirements.

10.4.2 Admin Template Method Requirements

There are no Admin Template method requirements.

10.5 Locking Template

This subsection defines the modified Locking Template as applicable for this specification.

10.5.1 Locking Template Table Requirements

The Locking Template table access requirements in Table 18 are mandatory.

Table 18 Locking Template Tables

Table Name	Table Type
LockingInfo	Array
Locking	Object

10.5.2 Locking Template Method Requirements

The following Locking Template method requirements in Table 19 are mandatory.

Table 19 Locking Template Methods

Method Name	Method Type	Requirement
Erase	Object	Mandatory

10.5.3 Locking Template Table Details

10.5.3.1 K_AES Media Encryption Key Tables

This section identifies deviations from [2] regarding the tables used to store media encryption keys as required by this specification.

10.5.3.1.1 K_AES Media Encryption Key Table UIDs

This section defines the UIDs of the tables used to store media encryption keys.

Table 20 Media Encryption Key Table UIDs

UID of Table Object	UID of Table	Table Name	Template
00 00 00 01 00 00 08 05	00 00 08 05 00 00 00 00	K_AES_128	Locking
00 00 00 01 00 00 08 06	00 00 08 06 00 00 00 00	K_AES_256	Locking

10.5.3.1.2 Media Encryption Key Table Group - K_AES_128 (Object Table)

Table 21 K_AES_128 Table Description

Column	IsIndex	Type	Description
UID		uid	This is the unique identifier for this object. (Read-only)
Name	Yes	name	This is the name of this object. (Read-only for pre-personalization objects)
CommonName	Yes	common_name	A name that MAY be shared among multiple K_AES_128 objects (Read-only for prepersonalization objects)
Key		bytes{max=32}	Key
Mode		symmetric_mode	Defines the mode with which this key SHALL be used.

The `Mode` column defines the encryption mode with which this key SHALL be used. MediaEncryption mode permits a vendor-specific encryption mode. Any byte beyond the first 16 in the `Key` column SHALL be ignored for the ECB, CBC, CFB, OFB, GCM, CCM, CTR modes. For MediaEncryption mode, the content of the Key column MAY be vendor-specific.

10.5.3.1.3 Media Encryption Key Table Group - K_AES_256 (Object Table)

Table 22 K_AES_256 Table Description

Column	IsIndex	Type	Description
UID		uid	This is the unique identifier for this object. (Read-only)
Name	Yes	name	This is the name of this object. (Read-only for pre-personalization objects)
CommonName	Yes	common_name	A name that MAY be shared among multiple K_AES_256 objects (Read-only for prepersonalization objects)
Key		bytes{max=64}	Key
Mode		symmetric_mode	Defines the mode with which this key SHALL be used.

The `Mode` column defines the encryption mode with which this key SHALL be used. MediaEncryption mode permits a vendor-specific encryption mode. Any byte beyond the first 32 in the `Key` column SHALL be ignored for the ECB, CBC, CFB, OFB, GCM, CCM, CTR modes. For MediaEncryption mode, the content of the Key column MAY be vendor-specific.

10.5.3.2 symmetric_mode Type definition

The symmetric_mode type used in the Mode column of the K_* tables is an enumeration type that has the following values:

- 0 = ECB
- 1 = CBC
- 2 = CFB
- 3 = OFB
- 4 = GCM
- 5 = CTR
- 6 = CCM
- 7 = XTS
- 8 = LRW
- 9 = EME
- 10 = CMC
- 11 = XEX
- 12-22 = Reserved
- 23 = Media Encryption

10.5.4 Locking Template Method Details

10.5.4.1 Erase

The UID for the Erase method is: 00 00 00 06 00 00 08 03

The Erase method is specific for this specification and SHALL be supported by the device. This method is used to cryptographically erase user data within a specific LBA Range and to reset the access control (‘Locking”) of that LBA Range.

The Erase method is an object method and is defined as:

```
LockingObjectUID.Erase [ ]
=>
[ ]
```

When invoked, the method’s side effects are:

- The TPer SHALL eradicate the current data encryption key for the LBA Range managed by the Locking object on which the method is invoked;
- The TPer SHALL generate a new data encryption key for the LBA Range managed by the Locking object on which the method is invoked;
- The TPer SHALL reset the ReadLockEnabled, WriteLockEnabled, ReadLocked, and WriteLocked column values to “False” for the Locking object on which the method is invoked;
- The TPer SHALL set the associated BandMaster credential to the MSID Credential value (see 12) and, when applicable, set Tries to zero.

- The TPer SHALL NOT change `RangeStart` and `RangeLength`.

The method call fails with TCG status NOT_AUTHORIZED if:

- The referenced object does not exist;
- The referenced object is not an object stored in the `Locking` Table.

10.6 Crypto Template

This subsection defines the modified Crypto Template as applicable for this specification.

10.6.1 Crypto Template Table Requirements

There are no Crypto Template table requirements.

10.6.2 Crypto Template Method Requirements

The Crypto Template methods access requirements in Table 23 are mandatory.

Table 23 Crypto Template Methods

Method Name	Method Type
Random	SP

10.6.3 Crypto Template Method Details

10.6.3.1 Random

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

- Count

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

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

11 SP Implementation Details

11.1 SP life cycle

Enterprise SSC-compliant TPers in which SPs are created in manufacturing SHALL conform to the life cycle defined in this section.

SPs created in manufacturing in an Enterprise SSC-compliant TPer SHALL support the "Manufactured" state as defined in this specification. Other SP states and their state-specific implications are outside the scope of this specification and considered vendor specific.

The "Manufactured" state is the standard operational state of an SP created in the manufacturing process, which defines the initial required access control settings and preconfigurations of an SP based on the Templates incorporated into the SP, prior to personalization.

For Enterprise SSC-compliant TPers that support issuance, refer to [2] for the life cycle states and life cycle management. [2] describes the life cycle states for SPs that are created through the issuance process. TPer requirements attendant to issuance are outside the scope of this document.

11.2 General SP Details

This section defines specific requirements as applicable to both the Admin SP and the Locking SP.

11.2.1 Anybody Authority Deviations

This section identifies deviations from [2] regarding the operation of the Anybody Authority as required by this specification.

The Anybody authority has an `Operation` column value of None. Per [2], an authority with an `Operation` value of None MAY be parameterized during session startup as a signing authority (HostSigningAuthority in the `StartSession` method, or SPSigningAuthority – the `ResponseSign` column value of the Control Authority).

Also per[2] , parameterization of an authority with an `Operation` value of None as an exchange authority (HostExchangeAuthority in the `StartSession` method, or SPExchangeAuthority – the `ResponseExch` column value of the Control Authority) SHALL result in method failure.

Thus, during session startup, it is not an error for the Anybody authority to be parameterized as the HostSigningAuthority in the `StartSession` method. If the Anybody authority is submitted in this manner, any value submitted in the HostChallenge parameter SHALL be ignored and disregarded.

It is also not an error for a referenced HostSigningAuthority to have the Anybody authority as its `ResponseSign` column value.

The Anybody authority can be submitted as a parameter of the `Authenticate` method, either with or without a value in the `Authenticate` method's Challenge parameter. Any value submitted in the Challenge parameter when the Authority parameter is the Anybody authority SHALL be ignored and disregarded. Assuming all other `Authenticate` method syntax is correct, the method SHALL return a Success status and a True result.

The Anybody authority counts against the maximum number of authenticated authorities permitted per session (as reported in the `Properties` method response MaxAuthentications field). Thus, if the maximum number of authorities that MAY be authenticated within a session is 8, the Anybody authority counts as one of these, and the host MAY authenticate up to 7 additional authorities during session startup or using the `Authenticate` method.

11.2.2 Authenticate Method Deviations

This section identifies deviations from [2] regarding the operation of the `Authenticate` method as required by this specification.

Authorities that require symmetric key challenge/response authentication are identified by the value of their `Operation` column. Authorities that use SymK SHALL have an `Operation` column value of 4. Such an authority MAY be authenticated through the use of the `Authenticate` method, or via session startup. This authority MAY be used as the HostSigningAuthority in the `StartSession` method call, or it MAY be referenced from the Control Authority's `ResponseSign` column value as the SPSigningAuthority.

In addition to having an `Operation` column value of 4 ("SymK"), a SymK authority SHALL also have a `Credential` column value that is a valid uidref to a valid symmetric key credential object (for instance, a `C_AES_128` or `C_AES_256` object). That credential SHALL have a `Mode` column value of 0 ("ECB"). If such an authority is invoked as the Authority parameter of the `Authenticate` method, the result of the intial method invocation SHALL be a 32-byte nonce that is the challenge from the SP.

Begin Informative Note

The host is able to use the storage device's capability of generating highly random byte sequences to create keys or passwords. One way to achieve this is via the `Random` method. Another mechanism to achieve this is by invoking the `Authenticate` method with the MakerSymK within a session to the Admin SP; the storage device responds with 32 bytes of random data.

End Informative Note

11.3 Admin SP

This subsection defines specific requirements as applicable to the Admin SP.

The Admin SP SHALL instantiate the Base Template, Admin Template and Crypto Template subject to requirements in section 10 of this specification.

11.3.1 Authorities & Credentials

This section defines the authorities and credentials required in the Admin SP.

11.3.1.1 Authority Table

Table 24 Admin SP Authority table

UID	Name	Common Name	IsClass	Class	Enabled	Secure	HashAnd Sign	Present Certificate	
00 00 00 09 00 00 00 01	"Anybody"	"Anybody"	FALSE	Null	TRUE	None	None	FALSE	a
00 00 00 09 00 00 00 03	"Makers"	"Maker"	TRUE	Null	TRUE	None	None	FALSE	b

UID	Name	Common Name	IsClass	Class	Enabled	Secure	HashAnd Sign	Present Certificate
00 00 00 09 00 00 00 04	"MakerSymK"	"Maker"	FALSE	"Makers"	TRUE	None	None	FALSE
00 00 00 09 00 00 00 06	"SID"	"TPerOwner"	FALSE	Null	TRUE	None	None	FALSE

▲ c d

Continued

Operation	Credential	Response Sign	Response Exch	Clock Start	Clock End	Limit	Uses	Log	LogTo
None	Null	Null	Null	<date_0_value>	<date_0_value>	0	0	None	Null
None	Null	Null	Null	<date_0_value>	<date_0_value>	0	0	None	Null
SymK	VU[1]	Null	Null	<date_0_value>	<date_0_value>	0	0	None	Null
Password	00 00 00 0B 00 00 00 01	Null	Null	<date_0_value>	<date_0_value>	0	0	None	Null

▲ a b c d

[1] This column value SHALL be a reference to a valid symmetric key credential object.

11.3.1.2 Credential Table Group (C_PIN) table

The Admin SP C_PINs are defined inTable 25. PIN values are a maximum size of 32-bytes each.

Table 25 Admin SP C_PIN table

UID	Name	Common Name	PIN	CharSet	TryLimit	Tries	Persistence
00 00 00 0B 00 00 00 01	"SID"	""	<PIN0_value>	Null	VU	VU	VU
00 00 00 0B 00 00 84 02	"MSID"	""	<PIN1_value>	Null	VU	VU	VU

This specification defines a SSC specific non-changeable PIN known as MSID. This PIN value SHALL be set at manufacturing time and MAY be used as an initial value for other PIN authority credential values on the device.

The SID PIN value SHALL be equal to the MSID Credential value at manufacturing time. Refer to section 12.1 for an informal discussion on the role of MSID Credential.

11.3.1.3 ACE table

The ACE definitions in Table 26 are required for compliance to this specification. The text in parentheses is only to provide clarification.

Table 26 Admin SP ACE table

UID	Name	Common Name	Boolean Expr	RowStart	RowEnd	ColStart	ColEnd
00 00 00 08 00 00 00 01	"Anybody"	""	00 00 00 09 00 00 00 01 (Anybody)	Null	Null	""	""
00 00 00 08 00 00 00 03	"Makers"	""	00 00 00 09 00 00 00 03 (Makers)	Null	Null	""	""
00 00 00 08 00 00 02 01	"SID"	""	00 00 00 09 00 00 00 06 (SID)	Null	Null	""	""
00 00 00 08 00 00 8C 03	"SID_SetSelf"	""	00 00 00 09 00 00 00 06 (SID)	Null	Null	"PIN"	"PIN"

UID	Name	Common Name	Boolean Expr	RowStart	RowEnd	ColStart	ColEnd
00 00 00 08 00 00 8C 04	"MSID_Get"	""	00 00 00 09 00 00 00 01 (Anybody)	Null	Null	"PIN"	"PIN"
00 00 00 08 00 00 8C 05	"SID_Set Makers"	""	00 00 00 09 00 00 00 06 (SID)	Null	Null	"Enabled"	"Enabled"

11.3.2 AccessControl table

The access control definitions in Table 27 are required for compliance to this specification. The text in parentheses is presented only for clarification. The `CommonName` fields in Table 27 are only for clarification and MAY exceed the length limit for names.

Table 27 Admin SP AccessControl table

Row Number	UID	InvokingID	MethodID	CommonName	ACL	Log	AddACE ACL	RemoveACE ACL	GetACL ACL
VU	VU	00 00 00 00 00 00 00 01 (ThisSP)	00 00 00 06 00 00 00 0C (Authenticate)	Anybody-Authenticate-AdminSP	00 00 00 08 00 00 00 01 (Anybody)	None	Null	Null	00 00 00 08 00 00 00 01 (Anybody)
VU	VU	00 00 00 09 00 00 00 00 (Authority table)	00 00 00 06 00 00 00 08 (Next)	Makers-Next-Authority table	00 00 00 08 00 00 00 03 (Makers)	None	Null	Null	00 00 00 08 00 00 00 03 (Makers)
VU	VU	00 00 00 09 00 00 00 01 (Anybody Authority object)	00 00 00 06 00 00 00 06 (Get)	Anybody-Get-Anybody Authority Object	00 00 00 08 00 00 00 01 (Anybody)	None	Null	Null	00 00 00 08 00 00 00 01 (Anybody)
VU	VU	00 00 00 09 00 00 00 03 (Makers Authority object)	00 00 00 06 00 00 00 06 (Get)	Makers-Get-Makers Authority Object	00 00 00 08 00 00 00 03 (Makers)	None	Null	Null	00 00 00 08 00 00 00 03 (Makers)
VU	VU	00 00 00 09 00 00 00 06 (SID Authority object)	00 00 00 06 00 00 00 06 (Get)	SID-Get-SID Authority Object	00 00 00 08 00 00 02 01 (SID)	None	Null	Null	00 00 00 08 00 00 02 01 (SID)
VU	VU	00 00 00 0B 00 00 00 00 (C_PIN table)	00 00 00 06 00 00 00 08 (Next)	Makers-Next-C_PIN table	00 00 00 08 00 00 00 02 (Makers)	None	Null	Null	00 00 00 08 00 00 00 02 (Makers)

Row Number	UID	InvokingID	MethodID	CommonName	ACL	Log	AddACE ACL	RemoveACE ACL	GetACL ACL
VU	VU	00 00 00 0B 00 00 00 01 (SID C_PIN object)	00 00 00 06 00 00 00 07 (Set)	SID_SetSelf-Set-SID_C_PIN object	00 00 00 08 00 00 8C 03 (SID_SetSelf)	None	Null	Null	00 00 00 08 00 00 02 01 (SID)
VU	VU	00 00 00 0B 00 00 84 02 (MSID C_PIN object)	00 00 00 06 00 00 00 06 (Get)	MSID_Get-Get-MSID C_PIN Object	00 00 00 08 00 00 8C 04 (MSID_Get)	None	Null	Null	00 00 00 08 00 00 02 01 (SID)
VU	VU	00 00 00 09 00 00 00 03 (Makers Authority Object)	00 00 00 06 00 00 00 07 (Set)	SID_Set Makers - Set-Makers Authority Object	00 00 00 08 00 00 8C 05 (SID_Set Makers)	None	Null	Null	00 00 00 08 00 00 02 01 (SID)
VU	VU	00 00 00 00 00 00 00 01 (ThisSP)	00 00 00 06 00 00 06 01 (Random)	Anybody-Random	00 00 00 08 00 00 00 01 (Anybody)	None	Null	Null	00 00 00 08 00 00 00 01 (Anybody)

11.4 Locking SP

This subsection defines specific requirements for the Locking SP.

The SPID for the Locking SP is `00 00 02 05 00 01 00 01`.

The Locking SP SHALL instantiate the Base Template, Locking Template and Crypto Template subject to constraints set forth in section 10 of this specification.

11.4.1 Locking SP authorities

Each LBA range has an associated authority allowing an authorized entity to manage the associated LBA Range. Management includes locking and unlocking of that range, enabling or disabling range locking, and setting both start LBA and size of the range. The `Authority` table contains rows only for supported locking ranges.

Table 28 Locking SP Authority table

UID	Name	CommonName	IsClass	Class	Enabled	Secure	HashAndSign	PresentCertificate
00 00 00 09 00 00 00 01	"Anybody"	"Anybody"	FALSE	Null	TRUE	None	None	FALSE
00 00 00 09 00 00 80 01	"BandMaster0"	"BandMaster"	FALSE	00 00 00 09 00 00 84 03 (BandMasters)	TRUE	None	None	FALSE
00 00 00 09 00 00 80 02	"BandMaster1"	"BandMaster"	FALSE	00 00 00 09 00 00 84 03 (BandMasters)	TRUE	None	None	FALSE
•	•	•	•	•	•	•	•	•
00 00 00 09 00 00 84 00	"BandMaster1023"	"BandMaster"	FALSE	00 00 00 09 00 00 84 03 (BandMasters)	TRUE	None	None	FALSE

UID	Name	CommonName	IsClass	Class	Enabled	Secure	HashAndSign	Present Certificate
00 00 00 09 00 00 84 01	"EraseMaster"	"EraseMaster"	FALSE	Null	TRUE	None	None	FALSE
00 00 00 09 00 00 84 03	"BandMasters"	"BandMasters"	TRUE	Null	TRUE	None	None	FALSE

Continued

-	c	b	a	Operation	Credential	ResponseSign	ResponseExch	ClockStart	ClockEnd	Limit	Uses	Log	LogTo
•	Password	Password	None	None	Null	Null	Null	<date_0_value>	<date_0_value>	0	0	None	Null
•	00 00 00 0B 00 00 80 02 (BandMaster1 "PIN")	00 00 00 0B 00 00 80 01 (BandMaster0 "PIN")	Null	Null	Null	Null	Null	<date_0_value>	<date_0_value>	0	0	None	Null
•	Null	Null	Null	Null	Null	Null	Null	<date_0_value>	<date_0_value>	0	0	None	Null
•	Null	Null	Null	Null	Null	Null	Null	<date_0_value>	<date_0_value>	0	0	None	Null
•	Null	Null	Null	Null	Null	Null	Null	<date_0_value>	<date_0_value>	0	0	None	Null
•	Null	Null	Null	Null	Null	Null	Null	<date_0_value>	<date_0_value>	0	0	None	Null
•	Null	Null	Null	Null	Null	Null	Null	<date_0_value>	<date_0_value>	0	0	None	Null

	Operation	Credential	ResponseSign	ResponseExch	ClockStart	ClockEnd	Limit	Uses	Log	LogTo
d	Password	00 00 00 0B 00 00 84 00 (BandMaster 1023 "PIN")	Null	Null	<date_0_value>	<date_0_value>	0	0	None	Null
e	Password	00 00 00 0B 00 00 84 01 (EraseMaster "PIN")	Null	Null	<date_0_value>	<date_0_value>	0	0	None	Null
f	None	Null	Null	Null	<date_0_value>	<date_0_value>	0	0	None	Null

11.4.1.1 BandMaster0 Authority

BandMaster0 SHALL be associated with the Global Range; each additional Range SHALL have a dedicated BandMaster authority. The BandMaster authorities are defined in Table 28.

11.4.1.2 EraseMaster Authority

The EraseMaster authority is defined in Table 28.

Begin Informative Content

The EraseMaster is a single dedicated authority used to reset one or more LBA Ranges by invoking the `Erase` method on the Locking object representing that Range. This is typically done for repurposing the storage device or for recovery of a LBA Range for which the unlock credential value is lost or blocked.

End Informative Content

11.4.2 Credential Table (C_PIN)

The Locking SP C_PINs are defined in Table 29. PIN values are a maximum size of 32-bytes each. The C_PIN table contains rows only for supported locking ranges.

Table 29 Locking C_PIN table

UID	Name	CommonName	PIN	CharSet	TryLimit	Tries	Persistence
00 00 00 0B 00 00 80 01	"BandMaster0"	""	<PIN2_value>	Null	VU	VU	VU
00 00 00 0B 00 00 80 02	"BandMaster1"	""	<PIN3_value>	Null	VU	VU	VU
•	•	•	•	•	•	•	•
00 00 00 0B 00 00 84 00	"BandMaster1023"	""	<PIN1025_value>	Null	VU	VU	VU
00 00 00 0B 00 00 84 01	"EraseMaster"	""	<PIN1026_value>	Null	VU	VU	VU

The PIN value for all ranges SHALL be set to the MSID Credential value at manufacturing time.

11.4.3 Access Control Elements

The Locking SP Access Control Elements (ACEs) are defined in Table 30. The ACE table contains rows only for supported locking ranges.

Table 30 Locking SP ACE table

UID	Name	CommonName	BooleanExpr	RowStart	RowEnd	ColStart	ColEnd
00 00 00 08 00 00 00 01	"Anybody"	""	00 00 00 09 00 00 00 01 (Anybody)	Null	Null	""	""
00 00 00 08 00 00 80 01	"BandMaster0"	""	00 00 00 09 00 00 80 01 (BandMaster0)	Null	Null	""	""
00 00 00 08 00 00 84 01	"BandMaster0_ SetSelf"	""	00 00 00 09 00 00 80 01 (BandMaster0)	Null	Null	"PIN"	"PIN"
00 00 00 08 00 00 88 01	"BandMaster0_ SetBand"	""	00 00 00 09 00 00 80 01 (BandMaster0)	Null	Null	"ReadLockEnabled"	"LockOnReset"
00 00 00 08 00 00 80 02	"BandMaster1"	""	00 00 00 09 00 00 80 02 (BandMaster1)	Null	Null	""	""
00 00 00 08 00 00 84 02	"BandMaster1_ SetSelf"	""	00 00 00 09 00 00 80 02 (BandMaster1)	Null	Null	"PIN"	"PIN"

UID	Name	CommonName	BooleanExpr	RowStart	RowEnd	ColStart	ColEnd
00 00 00 08 00 00 8C 02	"EraseMaster_ SetSelf"	""	00 00 00 09 00 00 84 01 (EraseMaster)	Null	Null	"PIN"	"PIN"
00 00 00 08 00 00 8C 01	"EraseMaster"	""	00 00 00 09 00 00 84 01 (EraseMaster)	Null	Null	""	""
00 00 00 08 00 00 8C 00	"BandMaster1023_ SetBand"	""	00 00 00 09 00 00 84 00 (BandMaster1023)	Null	Null	"RangeStart"	"LockOnReset"
00 00 00 08 00 00 88 00	"BandMaster1023_ SetSelf"	""	00 00 00 09 00 00 84 00 (BandMaster1023)	Null	Null	"PIN"	"PIN"
00 00 00 08 00 00 84 00	"BandMaster1023"	""	00 00 00 09 00 00 84 00 (BandMaster1023)	Null	Null	""	""
•	•	•	•	•	•	•	•
00 00 00 08 00 00 88 02	"BandMaster1_ SetBand"	""	00 00 00 09 00 00 80 02 (BandMaster1)	Null	Null	"RangeStart"	"LockOnReset"

UID	Name	CommonName	BooleanExpr	RowStart	RowEnd	ColStart	ColEnd
00 00 00 08 00 00 8C 05	"AnyMaster"	""	00 00 00 09 00 00 84 03 or 00 00 00 09 00 00 84 01 (BandMasters or EraseMaster)	Null	Null	""	""
00 00 00 08 00 00 8C 06	"BandMasters"	""	00 00 00 09 00 00 84 03 (BandMasters)	Null	Null	""	""
00 00 00 08 00 02 00 01	"Anybody_ GetBand"	""	00 00 00 09 00 00 00 01 (Anybody)	Null	Null	"UID"	"ActiveKey"
00 00 00 08 00 03 BF FF	"Get_K_AES_Mode"	""	00 00 00 09 00 00 00 01 (Anybody)	Null	Null	"Mode"	"Mode"

11.4.4 AccessControl table

The Locking SP access control definitions in Table 31 are required for compliance to this specification. The text in parentheses is presented only for clarification. The `CommonName` fields are not required to be accessible by this specification and MAY exceed the length limit for names to clarify the access control. The `AccessControl` table contains rows only for supported locking ranges.

Table 31 Locking SP AccessControl table

RowNumber	UID	InvokingID	MethodID	CommonName	ACL	Log	AddACE ACL	RemoveACE ACL	GetACL ACL
VU	VU	00 00 00 00 00 00 00 01 (ThisSP)	00 00 00 06 00 00 00 0C (Authenticate)	Anybody-Authenticate-LockingSP	00 00 00 08 00 00 00 01 (Anybody)	None	Null	Null	00 00 00 08 00 00 00 01 (Anybody)
VU	VU	00 00 00 09 00 00 00 00 (Authority table)	00 00 00 06 00 00 00 08 (Next)	Anybody-Next-Authority table	00 00 00 08 00 00 00 01 (Anybody)	None	Null	Null	00 00 00 08 00 00 8C 05 (AnyMaster)
VU	VU	00 00 00 09 00 00 00 01 (Anybody Authority Object)	00 00 00 06 00 00 00 06 (Get)	Anybody-Get-Anybody Authority object	00 00 00 08 00 00 00 01 (Anybody)	None	Null	Null	00 00 00 08 00 00 00 01 (Anybody)
VU	VU	00 00 00 09 00 00 84 03 (BandMasters Authority Object)	00 00 00 06 00 00 00 06 (Get)	AnyMaster-Get-BandMasters Authority object	00 00 00 08 00 00 8C 05 (AnyMaster)	None	Null	Null	00 00 00 08 00 00 8C 05 (AnyMaster)
VU	VU	00 00 00 09 00 00 84 01 (EraseMaster Authority Object)	00 00 00 06 00 00 00 06 (Get)	EraseMaster-Get-Erase Authority object	00 00 00 08 00 00 8C 01 (EraseMaster)	None	Null	Null	00 00 00 08 00 00 8C 01 (EraseMaster)

RowNumber	UID	InvokingID	MethodID	CommonName	ACL	Log	AddACE ACL	RemoveACE ACL	GetACL ACL
VU	VU	00 00 00 09 00 00 80 01 (BandMaster0 Authority Object)	00 00 00 06 00 00 00 06 (Get)	BandMaster0-Get- BandMaster0 Authority object	00 00 00 08 00 00 80 01 (BandMaster0)	None	Null	Null	00 00 00 08 00 00 80 01 (BandMaster0)
.	.	.	.	.	.	None	.	.	.
VU	VU	00 00 00 09 00 00 84 00 (BandMaster1023 Authority Object)	00 00 00 06 00 00 00 06 (Get)	BandMaster1023-Get- BandMaster1023 Authority object	00 00 00 08 00 00 84 00 (BandMaster1023)	None	Null	Null	00 00 00 08 00 00 84 00 (BandMaster1023)
VU	VU	00 00 00 0B 00 00 00 00 (C_PIN table)	00 00 00 06 00 00 00 08 (Next)	AnyMaster- Next-C_PIN table	00 00 00 08 00 00 8C 05 (AnyMaster)	None	Null	Null	00 00 00 08 00 00 8C 05 (AnyMaster)
VU	VU	00 00 00 0B 00 00 84 01 (EraseMaster C_PIN object)	00 00 00 06 00 00 00 07 (Set)	EraseMaster_ SetSelf-Set-EraseMaster C_PIN object	00 00 00 08 00 00 8C 02 (EraseMaster_SetSelf)	None	Null	Null	00 00 00 08 00 00 8C 01 (EraseMaster)

RowNumber	UID	InvokingID	MethodID	CommonName	ACL	Log	AddACE ACL	RemoveACE ACL	GetACL ACL
VU	VU	00 00 00 0B 00 00 80 01 (BandMaster0 C_PIN object)	00 00 00 06 00 00 00 07 (Set)	BandMaster0_ SetSelf-Set-BandMaster0 C_PIN object	00 00 00 08 00 00 84 01 (BandMaster0_SetSelf)	None	Null	Null	00 00 00 08 00 00 80 01 (BandMaster0)
.	.	.	.	.	.	.	.	.	.
VU	VU	00 00 00 0B 00 00 84 00 (BandMaster1023 C_PIN object)	00 00 00 06 00 00 00 07 (Set)	BandMaster1023_SetSelf- Set-BandMaster1023 C_PIN object	00 00 00 08 00 00 88 00 (BandMaster1023_SetSelf)	None	Null	Null	00 00 00 08 00 00 84 00 (BandMaster1023)
VU	VU	00 00 08 01 00 00 00 00 (LockingInfo table row 1)	00 00 00 06 00 00 00 06 (Get)	Anybody- Get- LockingInfo table	00 00 00 08 00 00 00 01 (Anybody)	None	Null	Null	00 00 00 08 00 00 8C 05 (AnyMaster)
VU	VU	00 00 08 02 00 00 00 00 (Locking table)	00 00 00 06 00 00 00 08 (Next)	AnyMaster- Next- Locking table	00 00 00 08 00 00 8C 05 (AnyMaster)	None	Null	Null	00 00 00 08 00 00 8C 05 (AnyMaster)
VU	VU	00 00 08 02 00 00 00 01 (Global_Range Locking object)	00 00 00 06 00 00 00 07 (Set)	BandMaster 0_SetBand- Set-Global_Range Locking object	00 00 00 08 00 00 88 01 (BandMaster 0_SetBand)	None	Null	Null	00 00 00 08 00 00 80 01 (BandMaster0)

RowNumber	UID	InvokingID	MethodID	CommonName	ACL	Log	AddACE ACL	RemoveACE ACL	GetACL ACL
.	.	.	.	.	.	.	.	.	.
VU	VU	00 00 08 02 00 00 04 00 (Band1023 _ Locking object)	00 00 00 06 00 00 00 07 (Set)	BandMaster1023 _SetBand-Set-Band1023 _ Locking object	00 00 00 08 00 00 8C 00 (BandMaster1023 _SetBand)	None	Null	Null	00 00 00 08 00 00 84 00 (BandMaster1023)
VU	VU	00 00 08 02 00 00 00 01 (Global_Range Locking object)	00 00 00 06 00 00 00 06 (Get)	Anybody _GetBand-Get-Global_Range Locking object	00 00 00 08 00 02 00 01 (Anybody _GetBand)	None	Null	Null	00 00 00 08 00 00 80 01 (BandMaster0)
.	.	.	.	.	.	.	.	.	.
VU	VU	00 00 08 02 00 00 04 00 (Band1023 _ Locking object)	00 00 00 06 00 00 00 06 (Get)	Anybody _GetBand-Get-Band1023 _ Locking object	00 00 00 08 00 02 00 01 (Anybody_GetBand)	None	Null	Null	00 00 00 08 00 00 84 00 (BandMaster1023)

RowNumber	UID	InvokingID	MethodID	CommonName	ACL	Log	AddACE ACL	RemoveACE ACL	GetACL ACL
VU	VU	00 00 08 02 00 00 00 01 (Global_Range Locking object)	00 00 00 06 00 00 08 03 (Erase)	EraseMaster-Erase-Global_Range Locking object	00 00 00 08 00 00 8C 01 (EraseMaster)	None	Null	Null	00 00 00 08 00 00 80 01 (BandMaster0)
.	.	.	.	.	.	.	.	.	.
VU	VU	00 00 08 02 00 00 04 00 (Band1023 Locking object)	00 00 00 06 00 00 08 03 (Erase)	EraseMaster-Erase-Band1023 Locking object	00 00 00 08 00 00 8C 01 (EraseMaster)	None	Null	Null	00 00 00 08 00 00 84 00 (BandMaster1023)
VU	VU	00 00 80 01 00 00 00 00 (DataStore)	00 00 00 06 00 00 00 06 (Get)	Anybody-Get-DataStore	00 00 00 08 00 00 00 01 (Anybody)	None	Null	Null	00 00 00 08 00 00 00 01 (Anybody)
VU	VU	00 00 80 01 00 00 00 00 (DataStore)	00 00 00 06 00 00 00 07 (Set)	BandMasters-Set-DataStore	00 00 00 08 00 00 8C 06 (BandMasters)	None	Null	Null	00 00 00 08 00 00 8C 06 (BandMasters)
VU	VU	00 00 00 00 00 00 00 01 (ThisSP)	00 00 00 06 00 00 06 01 (Random)	Anybody-Random	00 00 00 08 00 00 00 01 (Anybody)	None	Null	Null	00 00 00 08 00 00 00 01 (Anybody)

RowNumber	UID	InvokingID	MethodID	CommonName	ACL	Log	AddACE ACL	RemoveACE ACL	GetACL ACL
VU	VU	00 00 08 05 00 00 00 01 (Global_Range_ AES_128)	00 00 00 06 00 00 00 06 (Get)	Anybody-Get- Global_Range_ AES128	00 00 00 08 00 03 BF FF (Get_K_AES_Mode)	None	Null	Null	00 00 00 08 00 00 00 01 (Anybody)
.	.	•	.	.	.	.	.	.	.
VU	VU	00 00 08 05 00 00 04 00 (Band1023_ AES_128)	00 00 00 06 00 00 00 06 (Get)	Anybody-Get- Band1023_ AES128	00 00 00 08 00 03 BF FF (Get_K_AES_Mode)	None	Null	Null	00 00 00 08 00 00 00 01 (Anybody)
VU	VU	00 00 08 06 00 00 00 01 (Global_Range_AES_256)	00 00 00 06 00 00 00 06 (Get)	Anybody-Get- Global_Range_ AES256	00 00 00 08 00 03 BF FF (Get_K_AES_Mode)	None	Null	Null	00 00 00 08 00 00 00 01 (Anybody)
.	.	•	.	.	.	.	.	.	.

RowNumber	UID	InvokingID	MethodID	CommonName	ACL	Log	AddACE ACL	RemoveACE ACL	GetACL ACL
VU	VU	00 00 08 06 00 00 04 00 (Band1023_AES_256)	00 00 00 06 00 00 00 06 (Get)	Anybody-Get-Band1023_AES256	00 00 00 08 00 03 BF FF (Get_K_AES_Mode)	None	Null	Null	00 00 00 08 00 00 00 01 (Anybody)

11.4.5 Locking Objects Definition

The LBA Range (“Locking”) objects are defined in Table 33. The `Locking` table SHALL at minimum have a single Locking object (“Global_Range”) and MAY contain one or more additional Locking objects. The implementation of greater than 1023 locking ranges is beyond the scope of this specification.

11.4.5.1 Locking Objects Deviations

This section identifies deviations from [2] regarding the operation of the `Locking` table as required by this specification.

11.4.5.1.1 Range Attributes

[2] identifies the Global Range as the first row of the `Locking` table. The `Locking` table is an object table, and row ordering is not defined for object tables. The Global Range is identifiable by its UID.

Additionally, the implied restriction that only the Global Range are able to have `RangeLength` and `RangeStart` both = 0 is also be lifted.

The restriction that the `RangeLength` and `RangeStart` columns of Locking objects other than the Global Range are unable to be changed after the row has been created is lifted, instead leaving that to the usual ACL control mechanism. Changes to the `RangeLength` and/or `RangeStart` columns are to be subjected to the same constraints and checks that are defined for those columns when rows of the locking table are created.

Locking objects whose `RangeLength` column has a value of 0 do not have any LBAs under their control and thus do not overlap any other row, even if their `RangeStart` values match. Any `Set` method invocation that results in a `Locking` Table row’s `RangeLength` column being non-zero, or that does not change a non-zero `RangeLength` column but does change a `RangeStart` column, is subject to the same overlapping range restrictions as already described in [2].

11.4.5.1.2 ReadLocked

The value of this column identifies the current read lock state for the associated LBA Range if the range's `ReadLockEnabled` column is True. This column is ignored if the range's `ReadLockEnabled` column is False. If this value is True, then the TPer shall not allow requests to read user data. If this value is False, then the TPer shall allow requests to read user data.

The `Set` method MAY be invoked by the host to change the value of this column and alter the read-lock state. Setting the column value to True read locks the range. Setting the column value to False read unlocks the range.

11.4.5.1.3 WriteLocked

The value of this column identifies the current write lock state for the associated LBA Range if the range's `WriteLockEnabled` column is True. This column is ignored if the range's `WriteLockEnabled` column is False. If this value is True, then the TPer shall not allow requests to write user data. If this value is False, then the TPer shall allow requests to write user data.

The `Set` method MAY be invoked by the host to change the value of this column and alter the write lock state. Setting the column value to True write locks the range. Setting the column value to False write unlocks the range.

11.4.5.1.4 Key Columns

The `ActiveKey` and `NextKey` columns are modified to only allow pointing to `K_AES_128` and `K_AES_256` tables.

The type of the `ActiveKey` and `NextKey` columns is changed to media_key_object_uidref. The definition of this type is presented inTable 32.

Table 32 mediakey_object_uidref type

ID	Name	Format	Description
00 00 00 05 00 00 10 03	mediakey_object_uidref	8	This is a reference type that SHALL be used specifically for uidrefs to media encryption key objects. When performing type checking, as part of that type checking the TPer SHALL validate that this uidref is to an objectin a media encryption key table

Table 33 Locking SP Locking table

UID	Name	CommonName	RangeStart	RangeLength	ReadLockEnabled	WriteLockEnabled	ReadLocked	WriteLocked	LockOnReset	
00 00 08 02 00 00 00 01	"Global_Range"	"Locking"	0	0	FALSE	FALSE	FALSE	FALSE	Power Cycle	a
00 00 08 02 00 00 00 02	"Band1"	"Locking"	0	0	FALSE	FALSE	FALSE	FALSE	Power Cycle	b
•	•	•	•	•	•	•	•	•	•	-
00 00 08 02 00 00 04 00	"Band1023"	"Locking"	0	0	FALSE	FALSE	FALSE	FALSE	Power Cycle	c

Continued

▶	ActiveKey [1]	NextKey	ReEncryptState	ReEncryptRequest	AdvKeyMode	VerfMode	ContOnReset	LastReEncryptLBA	LastReEncStat	GeneralStatus
a	00 00 08 05/6 00 00 00 01	VU	VU	VU	VU	VU	VU	VU	VU	VU
b	00 00 08 05/6 00 00 00 02	VU	VU	VU	VU	VU	VU	VU	VU	VU
-	•	•	•	•	•	•	•	•	•	•
c	00 00 08 05/6 00 00 04 00	VU	VU	VU	VU	VU	VU	VU	VU	VU

[1] UID of K_AES_128 or K_AES_256 table row for the range

The TPer SHALL support the `LockOnReset` column values of "[0]" ("Power Cycle") and "[ ]" (the empty set).

Begin Informative Content

Changing the size and/or location of LBA Ranges will result in loss of data.

End Informative Content

11.4.6 K_AES_128 Table

The encryption keys and mode are defined in Table 34 for Locking Objects using the AES 128 encryption algorithm.

Table 34 K_AES _128 table

UID	Name	CommonName	Key	Mode
00 00 08 05 00 00 00 01	"Global_Range-_AES_128"	""	VU	VU
00 00 08 05 00 00 00 02	"Band1 - AES_128"	""	VU	VU

UID	Name	CommonName	Key	Mode
•	•	•	•	•
00 00 08 05 00 00 04 00	"Band1023_ AES_128"	""	VU	VU

11.4.7 K_AES_256 Table

The encryption keys and mode are defined in Table 35 for Locking Objects using the AES 256 encryption algorithm.

Table 35 K_AES _256 table

UID	Name	CommonName	Key	Mode
00 00 08 06 00 00 00 01	"Global_Range- _AES_256"	""	VU	VU
00 00 08 06 00 00 00 02	"Band1 - AES_256"	""	VU	VU
•	•	•	•	•
00 00 08 06 00 00 04 00	"Band1023_ AES_256"	""	VU	VU

11.4.8 LockingInfo table

The LockingInfo information is defined in Table 36.

Table 36 LockingInfo table

Row Number	UID	Name	Version	Encrypt Support	MaxRanges	Max ReEncryptions	Keys AvailableCfg
1	00 00 08 01 00 00 00 01	VU	VU	Media Encryption	VU	VU	VU

11.4.9 DataStore table

The `DataStore` table provides a generic non-volatile storage in the TPer for host access and modification. TCG access control enforces write access authorization to only bandmasters but allows unconstrained read access.

Table UID: `00 00 80 01 00 00 00 00`

Name: “DataStore”

Type: Byte

Size: 1024 bytes min

Begin Informative Content

Use cases exist in which the host needs to store a limited amount of data in a TPer. Such a use case is one where the drive is moved between different hosts and the new host needs reference information to subsequently to get the drive unlock keys.

End Informative Content

The `DataStore` byte table is defined in Table 37.

Table 37 DataStore table

Row Number	Byte Value
0	Byte_0
•	•
n	Byte_n

Row addressing in the `DataStore` table begins at Row Number 0.

All bytes of the `DataStore` table SHALL be set to a value of `0x00` at manufacturing time.

11.4.10 Device Behavior Under Locking

The storage device SHALL terminate read commands that address consecutive LBAs in one or more LBA ranges for which `ReadLockEnabled`=True and `ReadLocked`=True.

The storage device SHALL terminate write commands that address consecutive LBAs in one or more LBA ranges for which `WriteLockEnabled`=True and `WriteLocked`=True.

When a command is terminated due to range locking, the storage device SHALL terminate the command with a “Data Protection Error” as defined in [7].

If the storage device receives a read or write command that addresses consecutive LBAs in more than one LBA range and the LBA ranges are not locked, the storage device SHALL either:

- Process the data transfer without an error,

or

- Terminate the command with “Other Invalid Command Parameter” as defined in [7].

The storage device's range crossing behavior SHALL be reported in Level 0 Discovery (see the 'Range Crosing' bit in section 3.6.2.7).

The device SHALL always abort the following commands:

For SCSI commands:

- READ LONG(10);
- READ LONG(16);
- WRITE LONG(10), (WR_UNCOR = 0);
- WRITE LONG(16), (WR_UNCOR = 0).

For ATA devices:

- READ LONG (obsolete);
- WRITE LONG (obsolete);
- SCT READ LONG;
- SCT WRITE LONG.

12 Appendix – MSID

12.1 Use of MSID

Begin Informative Content

The MSID Credential value is set at manufacturing time by the storage device vendor and is typically printed on the storage device label. It represents the device's initial storage device SID Credential value (owner's password) and is electronically readable over the interface by the host.

The MSID Credential value is used as an initial value for SID and an initial value for any BandMaster and EraseMaster Credentials on the Locking SP. During enrollment, the host reads the MSID value from the MSID Credential and uses that to authenticate and change all other Credential values on the storage device.

Having the MSID Credential value electronically available to the host constitutes a risk to the overall security of the device. It is therefore very important that the host executes a Take-Ownership scenario the moment a new device is inserted into the system, whereby it

- Invokes the `Erase()` method on every Range on the device;
- Replaces SID, any BandMaster, and EraseMaster Credentials with host known values.

If any of the above mentioned steps fails, then the host should reject the device from the system, as it could be compromised due to malicious behavior.

End Informative Content

13 Appendix –ParamCheck examples - Informative

13.1 Set Method Example

Using the LRC to protect a Set operation on a PIN

```
MSID_UID.Set[ Values = [ [(“PIN”, “ThisIsMyPin”)] ], ParamCheck = 0x2851]
```

Where ParamCheck is obtained by:

List of Values = [“ThisIsMyPin”]

ParamCheck = LRC(Pad(“ThisIsMyPin”))

= LRC (0x00 0x54 0x68 0x69 0x73 0x49 0x73 0x4D 0x79 0x50 0x69 0x6E)

= 0x2851

13.2 Get Method Example

Using the LRC when retrieving a PIN

```
MSID_UID.Get[[(“startCol”,“PIN”), (“endCol”,“PIN”)], ParamCheck = 1]
   =>
[
   [ [(“PIN”, “ThisIsMyPin”)] ],
   (ParamCheck, 0x2851)
]
```

Where the LRC value is calculated using the same procedure as in “Using the LRC to protect a Set operation on a PIN”.