TCG TSS 2.0 TAB and Resource Manager Specification

Family "2.0", Level 00
Version 1.0, Revision 18
April 2019

Contact: admin@trustedcomputinggroup.org

PUBLISHED

TCG

Disclaimers, Notices, and License Terms

Copyright Licenses:

- Trusted Computing Group (TCG) grants to the user of the source code in this specification (the “Source Code”) a worldwide, irrevocable, nonexclusive, royalty free, copyright license to reproduce, create derivative works, distribute, display and perform the Source Code and derivative works thereof, and to grant others the rights granted herein.
- The TCG grants to the user of the other parts of the specification (other than the Source Code) the rights to reproduce, distribute, display, and perform the specification solely for the purpose of developing products based on such documents.

Source Code Distribution Conditions:

- Redistributions of Source Code must retain the above copyright licenses, this list of conditions and the following disclaimers.
- Redistributions in binary form must reproduce the above copyright licenses, this list of conditions and the following disclaimers in the documentation and/or other materials provided with the distribution.

Disclaimers:

- THE COPYRIGHT LICENSES SET FORTH ABOVE DO NOT REPRESENT ANY FORM OF LICENSE OR WAIVER, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, WITH RESPECT TO PATENT RIGHTS HELD BY TCG MEMBERS (OR OTHER THIRD PARTIES) THAT MAY BE NECESSARY TO IMPLEMENT THIS SPECIFICATION OR OTHERWISE. Contact TCG Administration (admin@trustedcomputinggroup.org) for information on specification licensing rights available through TCG membership agreements.
- THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, ACCURACY, COMPLETENESS, OR NONINFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.
- Without limitation, TCG and its members and licensors disclaim all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

Any marks and brands contained herein are the property of their respective owners.

Corrections and Comments

Please send comments and corrections to admin@trustedcomputinggroup.org

Normative-Informative Language

“SHALL,” “SHALL NOT”, “SHOULD”, ”SHOULD NOT”, “RECOMMENDED”, “MAY” and “OPTIONAL” in this document are normative statements. They are to be interpreted as described in [RFC-2119].

Acknowledgements

TCG and the TSS Work Group would like to thank the following people for their work on this specification.

- Will Arthur Raytheon
- Brenda Baggaley OnBoard Security
- Dave Challener Johns Hopkins University, APL
- Mike Cox OnBoard Security
- Andreas Fuchs Fraunhofer SIT
- Ken Goldman IBM
- Jürgen Repp Fraunhofer SIT
- Philip Tricca Intel
- Lee Wilson OnBoard Security

Table of Contents

1 General Information on The TCG TSS 2.0 Specification Library

1.1 Acronyms

For definitions of the acronyms used in the TSS 2.0 specifications please see the TCG TSS 2.0 Overview and Common Structures Specification [22}.

1.2 TCG Software Stack 2.0 (TSS 2.0) Specification Library Structure

At the time of writing, the documents that are part of the specification of the TSS 2.0 are:

[1] TCG TSS 2.0 Overview and Common Structures Specification

[2] TCG TSS 2.0 TPM Command Transmission Interface (TCTI) API Specification

[3] TCG TSS 2.0 Marshaling/Unmarshaling API Specification

[4] TCG TSS 2.0 System API (SAPI) Specification

[5] TCG TSS 2.0 Enhanced System API (ESAPI) Specification

[6] TCG TSS 2.0 Feature API (FAPI) Specification

[7] TCG TSS 2.0 TAB and Resource Manager Specification

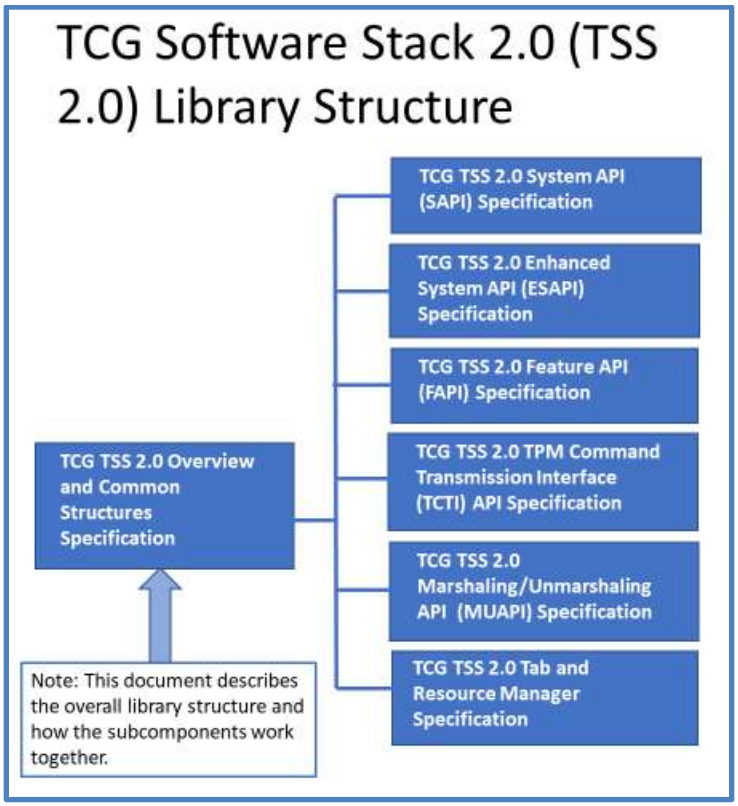

Figure 1: TSS 2.0 Specification Library

All references and acronyms for the TSS 2.0 library are in the TCG TSS 2.0 Overview and Common Structures Specification [22].

1.3 References

All references for the TSS 2.0 specifications are provided in the TCG TSS 2.0 Overview and Common Structures Specification [22].

2 Introduction

This document states the requirements for the TSS TAB (TPM access broker) and resource manager (RM) layers in the TSS software stack. These requirements are not APIs, but rather functional requirements that allow upper layers in the TSS software stack to interoperate. Another way of describing this is that these requirements define what the upper layers of the TSS can expect the TAB and resource manager to do for them.

2.1 TAB and Resource Manager Overview

The Resource Manager (RM) manages the TPM context in a manner similar to a virtual memory manager. it swaps objects, sessions, and sequences in and out of the limited TPM memory as needed. This layer is mostly transparent to the upper layers of the TSS and is not mandatory. However, if not implemented, the upper layers will be responsible for TPM context management.

The TPM access broker (TAB) handles multi-process synchronization to the TPM. A process accessing the TPM can be guaranteed that it will be able to complete a TPM command without interference from other competing processes.

2.2 Scope

The TAB/RM specification defines functional properties of a module that is expected to be implemented inside a system-wide scheduling daemon. The daemon could possibly implement other features such as access control restrictions. The internal interfaces between this daemon and the TAB/RM module are implementation specific and will therefore not be specified by TCG. Instead only the functional properties are specified and are expected to operate on TPM commands and responses and a cancel and set locality call.

A vendor of such a daemon is expected to provide a matched pair consisting of this daemon and a TCTI-Client-Provider. The interface between the TCTI and the RM is not specified by TCG.

In order to meet as many system requirements as possible, this specification attempts to avoid over-specifying and leaves lots of room for different implementations.

2.3 High Level Description

2.3.1 Multi-User Support

The TAB layer provides multi-user support to a single TPM device. The scheduling is per command from the caller. That is, there is no way for the caller to seize control of the TAB and send multiple consecutive commands with a guarantee that they won’t be intermixed with other caller's commands.

However, one caller command can map to multiple TPM commands, as described below. The TAB ensures that the TPM completes all commands related to a caller command before its scheduler services the next user.

NOTE Nothing in an implementation precludes layers of TABs. However, a TAB can assume that no other process at the same layer is connecting to a lower layer.

The TAB maintains a wait queue of caller commands to be scheduled. The TAB is permitted to (but not required to) optimize a command cancel request from the caller. If a command has not yet been sent to the TPM, the TAB can return a TPM_RC_CANCELED response and remove the command from the queue.

2.3.2 Application Transparency

This specification defines a resource manager (RM) that allows applications that run without a RM to also run with a RM, i.e. no code changes or recompilation are required in applications for the two environments.

For example, TPM2_ContextSave and TPM2_ContextLoad are allowed to be called from applications.

Full compatibility requires that applications be aware of and support the additional error codes required when a RM is present.

2.3.3 Context Swapping

The TPM is a limited resource device. It can hold a small number of objects and sessions. If more objects or sessions are used, they need to be swapped in and out of the TPM using the TPM2_ContextSave and TPM2_ContextLoad commands.

In a single process environment, a caller, perhaps the application, could handle the swapping. In a multi-process environment, the applications might be unaware of each other. A lower layer, called the resource manager (RM), handles this swapping.

The RM keeps a mapping of the following three data items:

- The TPM handle, returned by, and known to, the TPM
- The virtual handle (of transient objects), returned to, and known by, the caller
- The context, returned by TPM2_ContextSave and used by TPM2_ContextLoad

The RM deletes entries from its internal table(s) at several events.

The RM monitors TPM commands that flush objects or sessions from the TPM. For sessions, it monitors the response continueSession flag. It uses this information to manage the handle mapping and RM context storage. For example, it removes the mapping entry and deletes the context from its RM memory at the time of the flush.

The RM also monitors connections, and removes table entries and ContextFlushes leftover sessions when the caller closes a connection.

There are several implications, and they are different for objects and sessions.

2.3.3.1 *Objects*

For objects, context swapping can be performed transparently to the caller. However, each time a context is loaded, its TPM handle can change. To do this transparently, the RM returns a virtual handle, which does not change, to the caller. Each time the caller uses the object, it specifies this virtual handle. The RM performs a mapping from the (unchanging) virtual handle to the (currently assigned) TPM handle. It replaces the virtual handle with the TPM handle in the TPM command packet.

NOTE: The TPM 2.0 library specification excludes the handle from command stream HMAC calculations to enable this substitution.

Since there is no TPM limit on the number of saved object contexts, the only limit imposed on the RM is based on the 2^24 handle name space per connection. RM implementations can choose smaller values.

2.3.3.2 *Sessions*

Since a session retains its handle through save/load context cycles, the RM does not virtualize session handles. The purpose of swapping sessions in and out of the TPM is to make the number of session slots transparent to the user, even though the maximum number of active sessions remains a global limit of the TPM.

The RM monitors TPM commands that flush sessions (which in this case includes responses with the continueSession flag clear), and uses that information to manage the mapping between handles and context storage.

The RM also handles the session context gap (see the TPM 2.0 library specification). Since the session handle does not change as sessions are cycled, the process of handling the session gap is transparent to the caller except for perhaps a delay in processing of TPM commands.

The context gap does have an important side effect that the RM needs to cope with when it supports a context save and load from a caller. First, the RM needs to keep a copy of the context for use during context gap processing. Second, the caller will be unaware that the saved context changed during context gap processing. Therefore, the context blob returned to the caller cannot be used directly. The RM keeps a mapping between the caller's context blob (which could, for example, contain only a nonce) and its local copy.

NOTE: For this reason, applications cannot rely on non-opaque fields of the returned context blob.

3 Design Requirements

The following functional requirements apply to the TAB and resource manager.

3.1 Returning TAB/RM Errors

TAB/RM internal errors MUST be returned to the SAPI layer formatted as an TPM error response packet with the appropriate error level as described in the TSS 2.0 Overview and Common Structures Specification.

For a TAB/RM error, it MUST return one of the following error code levels:

- TSS2_RESMGRTPM_RC_LAYER is an error level indicator used for returning TPM error codes from Part 2 of the TPM 2.0 specification as TAB/RM errors. This means that the least significant two bytes of the error code are decoded like TPM error codes.
- TSS2_RESMGR_RC_LAYER is used to return TAB/RM specific errors. The least significant two bytes are interpreted following the scheme for base error codes in TSS 2.0 Overview and Common Structures Specification .

The TAB/RM error code levels are defined as:

```
#define TSS2_RESMGRTPM_RC_LAYER       ( 11 << TSS2_RC_LAYER_SHIFT)
#define TSS2_RESMGR_RC_LAYER          ( 12 << TSS2_RC_LAYER_SHIFT)
```

NOTES:

1. For example, if the TAB/RM can't find an entry in its table for a virtual handle, it could return TPM_RC_HANDLE with the error level set to TSS2_RESMGRTPM_RC_LAYER (to indicate that the error was returned from the Resource Manager) and the correct encoding of the other bits to indicate which handle failed. Or, it could return an implementation-specific error code with the error level set to TSS2_RESMGR_RC_LAYER.

2. An implementation can determine what type of error level and error code to return for each instance of an error. The two error levels described above tell the caller how to interpret the error code.

3.2 Handling TPM2 Response Codes

In all cases not described below the RM MUST return the response code from a command to the caller unfiltered.

3.2.1 TPM2_RC_RETRY & TPM2_RC_TESTING

The RM MAY choose to retry commands in response to these response codes on behalf of the caller. As these RCs indicate that the command was not started, the TAB/RM MAY submit other commands for execution before the command that produced the RC is resubmitted. Implementations should be careful to not create a situation where the TAB/RM could get stuck retrying a command indefinitely. TAB/RM implementations SHOULD establish a finite limit on the number or duration of retries that will be made before the RC is sent back to the caller.

3.2.2 TPM2_RC_YIELDED

A TAB/RM MAY choose to resubmit commands on behalf of the caller when this response code is returned. According to the section 38 of the TPM2 architecture specification (part 1), executing another command ahead of the resubmission of the command that caused this RC may result in the loss of state associated with the command. A TAB/RM reordering command execution as an optimization SHOULD prioritize resubmission of the command that caused the TPM2_RC_YIELD over new commands up to some limit before sending the RC back to the caller.

3.2.3 TPM2_RC_NV_RATE

A TAB/RM MAY choose to resubmit commands on behalf of the caller when this response code is returned. Once this RC has been received the TAB/RM may choose to resubmit the command with some delay. A TAB/RM that reorders command execution MAY chose to use the TPM2_RC_NV_RATE RC as input into its scheduling algorithm. Commands that don't interact with NV are not impacted by the rate limit and the TAB/RM may choose to schedule them ahead of resubmitting commands in response to this RC.

3.3 Cancel, setLocality, and makeSticky Commands

The cancel, setLocality, and makeSticky commands, if they are implemented in the TCTI layer above the TAB/RM, SHALL be handled appropriately by the TAB/RM. Specifically:

- Localities SHALL be maintained on a per-connection basis. The TAB/RM, before sending a command to the TPM SHALL switch the TPM to the locality last set by the connection via the TCTI setLocality function.
- Cancel command:
    - If sent via a connection the cancel command SHALL affect only a TPM command sent over the same connection.
    - If a resource manager removes a command from the wait queue when a cancel is received, it SHALL send back a TPM response message on the corresponding connection with return code TPM_RC_CANCELED and the level indicator set to TSS2_RESMGRTPM_RC_LAYER.
- makeSticky command:
    - If 'sticky' is 1, the RM SHALL ensure that the session, sequence or object identified by a virtual handle is loaded in TPM memory and return the real handle. Making a handle sticky means that it will stay loaded in TPM memory, meaning the RM won't unload it. This allows another process (typically a kernel) to use the object without concern about whether it's loaded in TPM memory.
    - If 'sticky' is 0, the RM SHALL resume its normal management of the session, sequence, or object identified by a real handle and return the virtual handle. Making a handle unsticky means that the RM is free to manage it as it desires.

- The RM SHALL ensure that if a session, sequence, or object is made sticky, it is only made unsticky by the same connection.
- When the connection is terminated, the RM SHALL flush all sticky sessions, sequences, and objects from the TPM.

NOTES:

1. Due to the nature of this function - being able to do a denial of service attack on the TPM - it typically will be restricted for usage by privileged applications and/or users only. Details on how to identify those or what privileged means are platform and stack specific and thereby out of scope for this specification

2. In order to use a trusted key ring from the Linux kernel, an application must be able to pass a real handle for a loaded object to the kernel. It can't pass a virtual handle because the kernel is bypassing the RM which means that no virtual to physical handle translation is possible.

- When a session, sequence, or object is sticky, the RM SHALL ensure that its previous virtual handle can't be used to refer to the session, sequence, or object.

NOTE: For sessions this is required to avoid application/kernel conflicts because session state changes whenever it's used. For sequences and objects, this isn't strictly required, but it was specified this way for the sake of consistency.

3.4 Per-connection Limits

A TAB/RM MAY enforce a maximum number of session and/or object and sequence handles per connection and MAY guarantee a minimum number of such handles per connection. This maximum MAY be variable for different connections. The TAB MAY also limit the number of concurrent connections in order to guarantee minimums to each connection.

NOTE: One reason a RM may allow variable maximums would be to support different levels of privilege.

A TAB/RM SHOULD not limit the number of concurrent connections.

3.5 Priority

The TAB/RM scheduling algorithm is implementation defined.

The algorithm could be a round robin or FIFO, with no priority scheduling. Command reordering could give priority to short commands.

If privilege is implemented, priority may be given to privileged connections.

3.6 Privilege

This version of the specification does not address privileged connections.

A future version may require privileged connections. Connections can have attributes such as:

- priority in scheduling

- different access to commands

For example, a privileged connection could permit a TPM2_PCR_Extend to PCR 0, while that command would be blocked for unprivileged connections. A privileged connection would be less likely to have a session flushed. Commands with high impact, such as TPM2_Clear could be restricted.

3.7 Concurrency

TAB MUST guarantee that between the time the TAB sends a command to the TPM and gets that command's response from the TPM that no other TPM commands are sent to the TPM and no responses are allowed to be read by a different connection.

During this interval, other commands sent to the TAB/RM MUST be queued up by the TAB/RM.

3.8 Isolation

Transient object and session handles MUST be isolated to the connection that created them. The TAB MUST prevent connections from accessing handles and objects that belong to other connections. All other types of handles MUST available to all connections.

A resource manager MUST allow TPM2_ContextSave and TPM2_ContextLoad commands from connections. Any object, sequence or session context that has been saved by a command through a connection MUST be loadable by any connection. This mechanism is intended for use as a way to pass ownership of objects and sessions between connections.

3.9 Before Sending a Command

The RM MUST ensure that all session, object, and sequence contexts required for execution of that command are context loaded into the TPM before the command byte stream is sent to the TPM.

3.10 Handling for Transient Objects and Sequences

For objects and sequences the following MUST be performed:

- The TAB/RM MUST maintain a logical mapping between RM assigned virtual handles, TPM real handles, and saved contexts.
- The RM MUST virtualize the handles:
    - In responses that return handles, these handles MUST be virtualized before returning the response to the caller:
        - These virtual handles MUST be unique per connection.
        - The RM MUST replace the real handles in the response with virtual handles.
        - Virtual handle substitutions in responses MUST preserve the most significant byte from the returned real handles since this contains the handle type, and layers of

software above the TAB/RM sometimes need this handle type.
NOTE: The least significant three bytes of virtual handles are implementation-specific.

- In commands, virtualized handles received from the caller MUST be replaced in the command byte stream with real handles before sending the command to the TPM.

- A TPM2_ContextSave command MUST appear to be executed normally and the actual object context returned to the caller

NOTE: For objects (not sequences), loaded object contexts never change, so RM could just save context once.

- For transient objects, flushing of a context may return TPM_RC_SUCCESS even when the TPM would return an error. This could occur in the following scenario:
  If one TCTI context, "A", has created a transient object in the owner or endorsement hierarchy that is currently swapped out in the RM and another TCTI context, 'B', performs a TPM2_Clear command, then a TPM2_FlushContext command executed by TCTI context A, would return TPM_RC_HANDLE if sent directly to the TPM. An optimized resource manager, which just deletes the transient object's record instead of doing a TPM2_ContextLoad followed by a TPM2_ContextFlush may return success.

3.11 Persistent Objects

If the underlying TPM requires a transient object slot to be available when a command uses a persistent object (as indicated by the TPMA_MEMORY objectCopiedToRam bit), then the RM MUST ensure that a transient object slot is available for such commands.

3.12 Invalid Contexts

Whenever a session, transient object, or sequence context is no longer valid, the TAB/RM SHOULD remove its handle to context mapping as well as the saved context itself (garbage collection).

NOTE: some examples of this are:

- For sessions: when a session is flushed or when the command's response has the continueSession bit cleared.
- For transient objects and sequences, a TPM2_FlushContext command from a connection typically causes the RM to remove its handle to context mapping and associated context (garbage collection).
- For transient objects, after a TPM Resume or a TPM Restart, if the object's stClear bit is set.
- For sequences, when the TPM2_SequenceComplete or TPM2_EventSequenceComplete commands successfully execute.
- If TPM Reset occurs, all transient objects and sequences are invalidated.

3.13 Connection Shutdown

When the TAB/RM discovers that a connection is shut down, e.g. via the TCTI's finalize call or an application exits or crashes:

- The RM SHOULD flush any objects, sequences, or sessions owned by the connection and remove any entries about these in its internal tables.
- The RM MAY send a TPM cancel command if the TPM is busy running a command for the connection that shut down.
- If the RM sends the cancel command, it MUST receive the TPM response. This response is never sent to the caller since the connection is being shut down.

3.14 Management of Sessions

The RM MUST not virtualize session handles.

NOTES:

- GetSessionAuditDigest and all the policy commands take a session handle in the handle area. In order to calculate the cpHash for these commands, the name of the session, which is the session handle, must be known to the application. For that reason, virtualization of session handles cannot be supported.
- A TPM2_ContextSave command MUST appear to be executed normally and the actual object context returned to the caller
- When a command uses sessions, this could require the RM to TPM2_ContextLoad a saved session context. During this operation a TPM_RC_CONTEXT_GAP error could be returned by the TPM. The RM MUST either proactively prevent this error or reactively correct the error by un-gapping the stored session contexts, then load the session, and perform the requested command. Some specific requirements for handling the gap scheme correctly are:
  - When a connection sends a TPM2_ContextSave command, the RM MUST save the context, so that it can context load and save (un-gap) it if this context is the reason for a gap response on a different session's TPM2_ContextLoad.
  - When a context that has been un-gapped by the RM is to be loaded via TPM2_ContextLoad from a connection (using the previously returned context blob) the RM SHALL substitute the passed in context with the un-gapped context saved by the resource manager.
- On a TPM2_ContextSave from a connection, the RM could choose to return a TPM2B placeholder as the contextBlob field inside the returned TPMS_CONTEXT. The reason for doing this would be to conserve memory—without this, the whole saved context plus the most recent context (in the event of a gap event) must be saved. For this reason, applications cannot rely on non-opaque fields of the returned context blob.
- TPM2_StartAuthsession MUST never fail due to TPM_RC_CONTEXT_GAP. The RM MUST either proactively prevent this error or reactively correct the error by un-gapping the stored session contexts (and then retrying the command).
- TPM2_StartAuthSession can fail with TPM_RC_SESSION_HANDLES even when a single connection has not yet exceeded its limits, because other connections may have allocation sessions that drain the pool of max active sessions of the underlying TPM. To prevent this, the RM may impose limits, see Section 3.4

3.15 Get Capability

In order for applications to be able to run either with or without a resource manager, some TPM capabilities are virtualized. In this way, regardless of whether a resource manager is present, applications get a proper view of the available capabilities.

When audit sessions are used, the RM SHALL NOT alter the response to TPM2_GetCapability commands. This is because the responses of the TPM are recorded in the audit digest of the session. Therefore, in order to keep the audit digest consistent, the responses are not altered if an audit session is present.

When audit sessions aren't used, TPM2_GetCapability commands for the following capability/property pairs SHALL return the following resource manager's versions of these values. All other values SHALL be passed through the RM unaltered.

When audit session aren't used, the following sub-sections define the expected behavior of the RM. Each section

NOTE:

1. When the following text refers to TPM_PT_XXX, it means the value returned by GetCapability for that property.
2. Because TPM2_GetCapability commands with audit sessions will return the TPM's version of these properties, audit sessions cannot be used by applications for TPM2_GetCapability commands..

3.15.1 List of Loaded Handles

If GetCapability is called with the capability and property fields set to TPM_CAP_HANDLES and a handle in the transient range, the RM SHALL return the list of exactly those handles of transient objects associated with the current connection.
NOTE: This includes virtualizes objects as well.

3.15.2 List of Loaded Sessions

If GetCapability is called with the capability and property fields set to TPM_CAP_HANDLES and the sessio n handle range, the RM SHALL return the list of exactly those handles of sessions associated with the current connection.

3.15.3 Number of Transient Objects in TPM RAM

If GetCapability is called with the capability and property fields set to TPM_CAP_TPM_PROPERTIES and TPM_PT_HR_TRANSIENT_MIN, the RM SHALL return an implementation-specific number in the range from the value as specified by the platform specification up to 0xffffff. The platform specification used is determined by the TPM_PT_FAMILY_INDICATOR, TPM_PT_LEVEL, and TPM_PT_REVISION values. The TPM_PT_HR_TRANSIENT_MIN property SHALL be connection-specific and constant over the lifetime of the connection.

The number of handles for transient objects per connection SHALL NOT be greater than the virtualized version of TPM_PT_HR_TRANSIENT_MIN.

NOTE:

1. For the TPM this is a minimum, but for the RM it's both a minimum and a maximum for the maximum number of loaded objects.

3.15.4 Number of Sessions in TPM RAM

If GetCapability is called with the capability and property fields set to TPM_CAP_TPM_PROPERTIES and TPM_PT_HR_LOADED_MIN, the RM SHALL return an implementation-specific number in the range from the value as specified by the platform specification up to the value for virtualized TPM_PT_ACTIVE_SESSIONS_MAX (see section 3.15.5). The platform specification used is determined by

the TPM_PT_FAMILY_INDICATOR, TPM_PT_LEVEL, and TPM_PT_REVISION values. The TPM_PT_HR_LOADED_MIN property SHALL be connection-specific and constant over the lifetime of the connection. The resource manager SHOULD return the virtualized TPM_PT_ACTIVE_SESSIONS_MAX value.

NOTE:

1. For the TPM, TPM_PT_HR_LOADED_MIN is the minimum number of authorization sessions. For the RM TPM_PT_HR_LOADED_MIN is the number of loaded sessions that are allowed per connection. It is recommended that the RM set this to the TPM_PT_HR_LOADED_MIN value from the appropriate TCG platform specification (e.g. TCG PC Client Specific Implementation Specification).

3.15.5 Maximum Number of Active Sessions

If GetCapability is called with the capability and property fields set to TPM_CAP_TPM_PROPERTIES and TPM_PT_ACTIVE_SESSIONS_MAX, the RM SHALL return an implementation-specific number in the range from the value of TPM_PT_HR_LOADED_MIN as specified by the platform specification up to the TPM returned value. The platform specification used is determined by the TPM_PT_FAMILY_INDICATOR, TPM_PT_LEVEL, and TPM_PT_REVISION values. The TPM_PT_ACTIVE_SESSIONS_MAX property SHALL be connection-specific and constant over the lifetime of the connection.

NOTE:

1. Even if this number is larger than the value of the platform specification's TPM_PT_HR_LOADED_MIN, applications are normally supposed to use the platform specification's TPM_PT_HR_LOADED_MIN value when deciding how many sessions to create in order to prevent running into the TPM's limit on the number of loaded sessions.

3.15.6 Context Gap

If GetCapability is called with the capability and property fields set to TPM_CAP_TPM_PROPERTIES and TPM_PT_CONTEXT_GAP_MAX the RM SHALL return 0xffffffff**.**

NOTE: This value indicates the presence of an RM that handles un-gapping schemes. If such a value is detected, then existing un-gapping code in any layer above the RM can possibly be deactivated.

3.15.7 TPM Persistent Memory Management

If GetCapability is called with the capability and property fields set to TPM_CAP_TPM_PROPERTIES and TPM_PT_MEMORY the RM SHALL alter the TPM's returned value by setting objectCopiedToRam to CLEAR. See section 3.11.

3.15.8 Session Context Sizes

If GetCapability is called with the capability and property fields set to TPM_CAP_TPM_PROPERTIES and TPM_PT_MAX_SESSION_CONTEXT and returned contexts are altered by the RM as described in Note 1 in section 3.14, RM SHALL return the size of the altered contexts. Otherwise, it SHALL return the size as returned by the TPM.

3.15.9 Number of TPM Loaded Sessions

If GetCapability is called with the capability and property fields set to TPM_CAP_TPM_PROPERTIES and TPM_PT_HR_LOADED the RM SHALL return the number of sessions that are loaded from the connection's point of view (see section 3.15.2):

- This value is incremented every time the connection executes TPM2_StartAuthSession or TPM2_ContextLoad on a session.
- This value is decremented every time a connection does TPM2_ContextSave, TPM2_FlushContext or continueSession is set to CLEAR.
- This value MUST NOT be greater than TPM_PT_HR_LOADED_MIN.

3.15.10 Number of Available TPM Session Slots

If GetCapability is called with the capability and property fields set to TPM_CAP_TPM_PROPERTIES and TPM_PT_HR_LOADED_AVAIL the RM SHALL return the difference:

(TPM_PT_HR_LOADED_MIN - TPM_PT_HR_LOADED)

This is connection-specific.

3.15.11 Number of Active Sessions

If GetCapability is called with the capability and property fields set to TPM_CAP_TPM_PROPERTIES and TPM_PT_HR_ACTIVE the RM SHALL return the number of sessions in the RM's table:

- This value is incremented every time the connection executes TPM2_StartAuthSession on a session.
- This value is decremented every time connection executes TPM2_Flush or continueSession is set to CLEAR.
- This value SHALL NOT be greater than TPM_PT_HR_ACTIVE_SESSIONS_MAX.
- The value MUST be derived using the following equation:
  TPM_PT_HR_ACTIVE = TPM_PT_ACTIVE_SESSIONS_MAX - TPM_PT_HR_ACTIVE_AVAIL

This is connection-specific.

3.15.12 Number of Available Transient Object Slots

If GetCapability is called with the capability and property fields set to TPM_CAP_TPM_PROPERTIES and TPM_PT_HR_TRANSIENT_AVAIL the RM SHALL return the difference:

- (TPM_PT_HR_TRANSIENT_MIN - "the number of objects in RM table")
  NOTE: see section 3.15.1.
- The RM SHALL NOT allow the creation of more concurrent objects than TPM_PT_HR_TRANSIENT_MIN.

This is connection-specific.

3.16 Handling for Vendor-Specific Commands

For vendor specific commands, the Resource Manager SHALL retrieve the number of handles in the handle area of the command by querying TPM_CAP_COMMANDS using the TPM2_GetCapabiliy command and handle them accordingly.

NOTE:

1. If handles are embedded within the parameter area there is no standard way to detect them. Accordingly, a Resource Manager can handle these special cases if it is aware of those. However, TPM-Vendors are strongly advised to only include handles within the handle area for vendor specific commands.

4 Non-Design Requirements:

The format of the RM's internal data structures used to maintain data pertaining to each virtual handle is implementation-specific.

5 Environmental Restrictions:

The following are some restrictions that must be maintained by the system in order for the TAB/RM to work properly.

- TAB/RM is assumed to be the only regular entry point by which applications talk to the TPM device driver.