# TPM Main
# Part 3 Commands

**Specification Version 1.2**
**Revision 62**
**2 October 2003**
**Published**

Contact: tpmwg@trustedcomputinggroup.org

**TCG PUBLISHED**



TCG

**Acknowledgement**

TCG wishes to thank all those who contributed to this specification. This version builds on the work published in version 1.1 and those who helped on that version have helped on this version.

A special thank you goes to the members of the TPM workgroup who had early access to this version and made invaluable contributions, corrections and support.

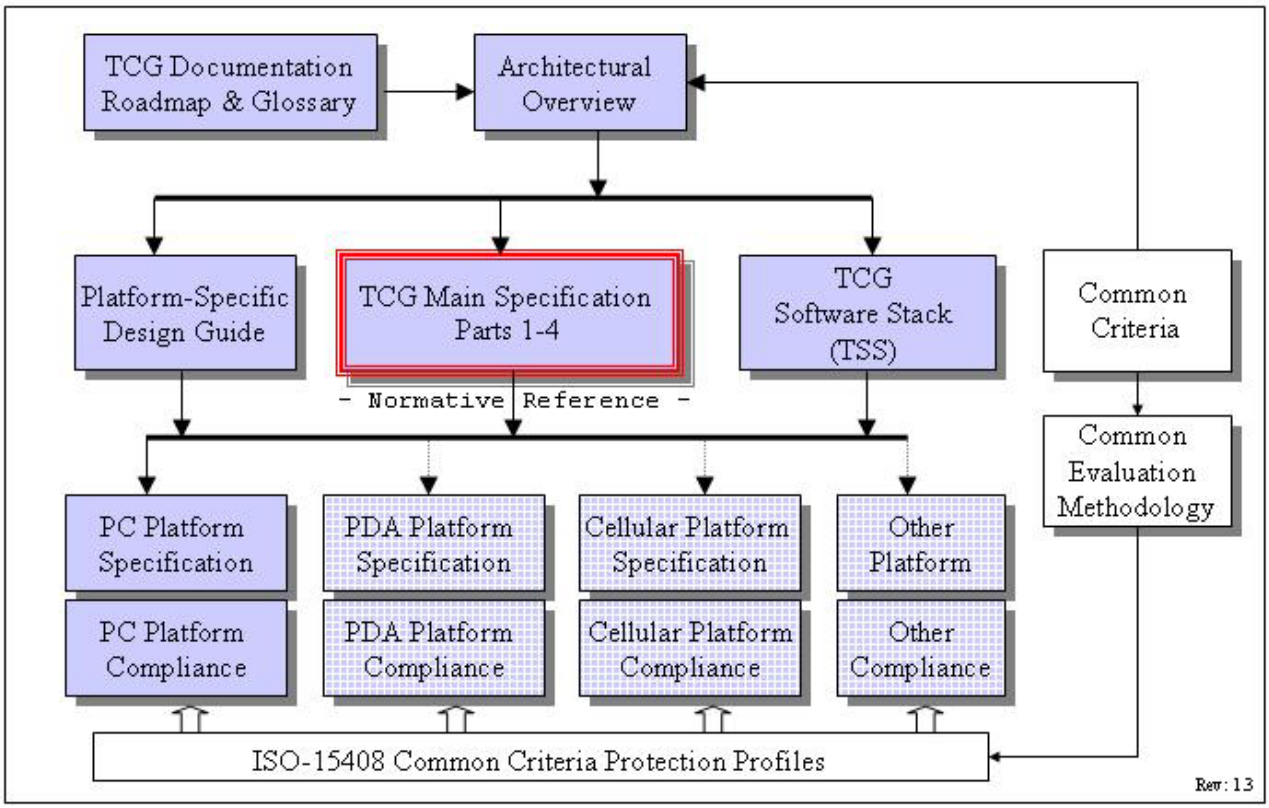
David Grawrock

TPM Workgroup chair

**Change History**

| Version | Date | Description |
|---|---|---|
| Rev 50 | Jul 2003 | Started 01 Jul 2003 by David Grawrock<br>Breakup into parts and the merge of 1.1 commands |

# TCG Doc Roadmap – Main Spec



# TCG Main Spec Roadmap

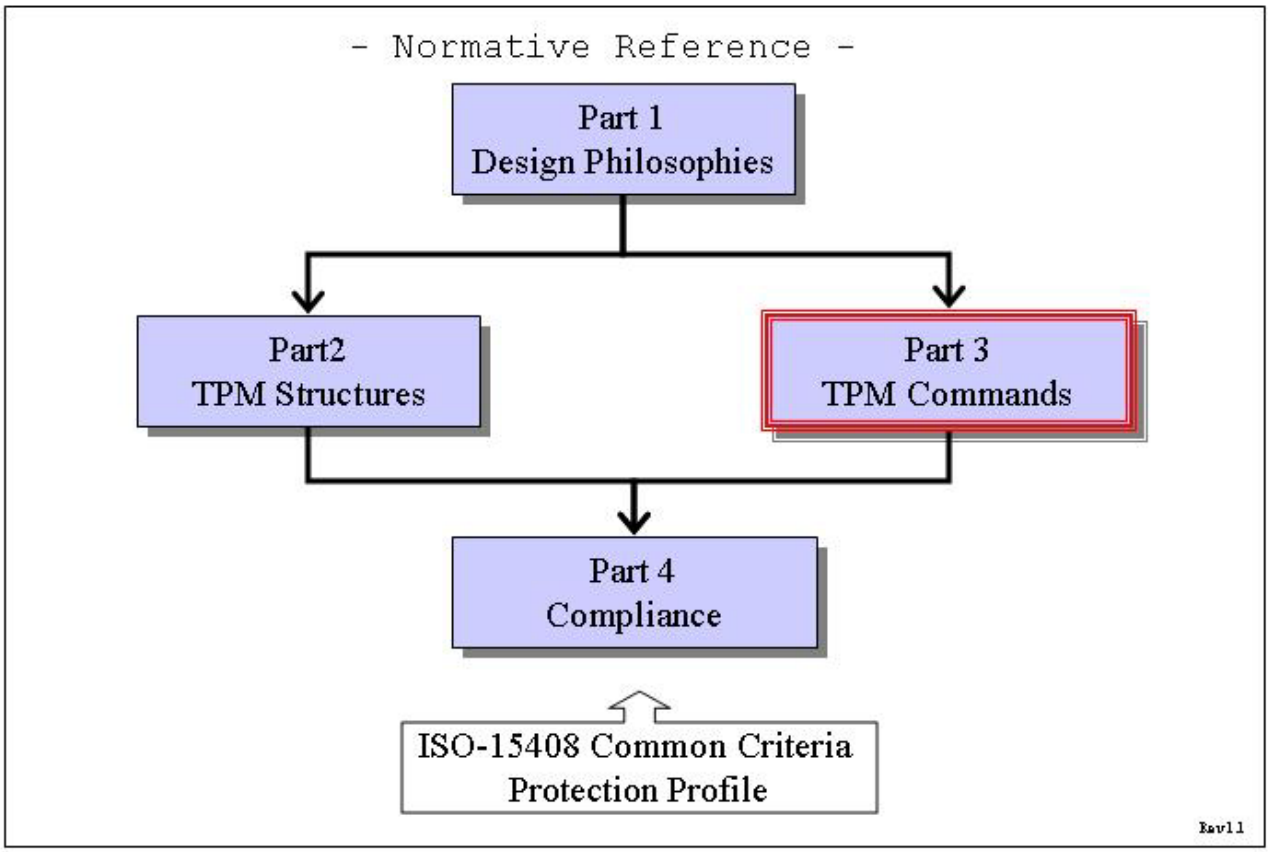


**Table of Contents**

End of Introduction do not delete

# 1. Scope and Audience

The TPCA main specification is an industry specification that enables trust in computing platforms in general. The main specification is broken into parts to make the role of each document clear. A version of the specification (like 1.2) requires all parts to be a complete specification.

This is Part 3 the structures that the TPM will use.

This document is an industry specification that enables trust in computing platforms in general.

## 1.1 Key words

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in the chapters 2-10 normative statements are to be interpreted as described in [RFC-2119].

## 1.2 Statement Type

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

For example:

***Start of informative comment:***

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

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

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

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

***End of informative comment***.

This is the first paragraph of one or more paragraphs (and/or sections) containing the text of the kind *normative statements* ...

To understand the TPM specification the user MUST read the specification. (This use of MUST indicates a keyword usage and requires an action).

# 2. Description and TODO

This document is to show the changes necessary to create the 1.2 version of the TCG specification. Some of the sections are brand new text; some are rewritten sections of the 1.1 version. Upon approval of the 1.2 changes, there will be a merging of the 1.1 and 1.2 versions to create a single 1.2 document.

# 3. Admin Startup and State

***Start of informative comment:***

This section is the commands that start a TPM.

***End of informative comment.***

## 3.1 TPM_Init

***Start of informative comment:***

TPM_Init is a physical method of initializing a TPM. There is no TPM_Init ordinal as this is a platform message sent on the platform internals to the TPM. On a PC this command arrives at the TPM via the LPC bus and informs the TPM that the platform is performing a boot process.

TPM_Init puts the TPM into a state where it waits for the command TPM_Startup (which specifies the type of initialization that is required.

***End of informative comment.***

**Definition**

```
TPM_Init();
```

**Type**

Operation of the TPM. This is not a command that any software can execute. It is inherent in the design of the TPM and the platform that the TPM resides on.

**Parameters**

None

**Description**

1. The TPM_Init signal indicates to the TPM that platform initialization is taking place. The TPM SHALL set the TPM into a state such that the only legal command to receive after the TPM_Init is the TPM_Startup command. The TPM_Startup will further indicate to the TPM how to handle and initialize the TPM resources.
2. The platform design MUST be that the TPM is not the only component undergoing initialization. If the TPM_Init signal forces the TPM to perform initialization then the platform MUST ensure that ALL components of the platform receive an initialization signal. This is to prevent an attacker from causing the TPM to initialize to a state where various masquerades are allowable. For instance, on a PC causing the TPM to initialize and expect measurements in PCR0 but the remainder of the platform does not initialize.
3. The design of the TPM MUST be such that the ONLY mechanism that signals TPM_Init also signals initialization to the other platform components.

**Actions**

1. The TPM sets TPM_STANY_FLAGS -> postInitialise to TRUE.

## 3.2 TPM_Startup

***Start of informative comment:***

TPM_Startup is always preceded by TPM_Init, which is the physical indication (a system-wide reset) that TPM initialization is necessary.

There are many events on a platform that can cause a reset and the response to these events can require different operations to occur on the TPM. The mere reset indication does not contain sufficient information to inform the TPM as to what type of reset is occurring. Additional information known by the platform initialization code needs transmitting to the TPM. The TPM_Startup command provides the mechanism to transmit the information.

The TPM can startup in three different modes:

A "clear" start where all variables go back to their default or non-volatile set state

A "save" start where the TPM recovers appropriate information and restores various values based on a prior TPM_SaveState. This recovery requires an invocation of TPM_Init to be successful.

A "deactivated" start where the TPM turns itself off and requires another TPM_Init before the TPM will execute in a fully operational state.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Startup |
| 4 | 2 | | | TPM_STARTUP_TYPE | startupType | Type of startup that is occurring |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Description**

TPM_Startup MUST be generated by a trusted entity (the RTM or the TPM, for example).

**Actions**

1. If TPM_STANY_FLAGS -> postInitialise is FALSE,
   a. Then the TPM MUST return TPM_INVALID_POSTINIT, and exit this capability
2. If stType = TPM_ST_CLEAR
   a. Ensure that sessions associated with resources TPM_RT_CONTEXT, TPM_RT_AUTH and TPM_RT_TRANS are invalidated

b. Reset PCR values to each correct default value
c. Set the following TPM_STCLEAR_FLAGS to their default state
    i. PhysicalPresence
    ii. PhysicalPresenceLock
    iii. disableForceClear
d. The TPM MAY initialize auditDigest to NULL
    i. If not initialized to NULL the TPM SHALL ensure that auditDigest contains a valid value
    ii. If initialization fails the TPM SHALL set auditDigest to NULL and SHALL set the internal TPM state so that the TPM returns TPM_FAILED_SELFTEST to all subsequent commands.
e. The TPM SHALL set TPM_STCLEAR_FLAGS -> deactivated to the same state as TPM_PERMANENT_FLAGS -> deactivated
f. The TPM MUST set the TPM_STANY_DATA fields to:
    i. TPM_STANY_DATA->contextNonceSession is set to NULLS
    ii. TPM_STANY_DATA->contextCount is set to 0
    iii. TPM_STANY_DATA->contextList is set to 0
g. The TPM MUST set TPM_STCLEAR_DATA fields to:
    i. Invalidate contextNonceKey
    ii. countID to NULL
    iii. bGlobalLock to FALSE
h. Determine which keys should remain in the TPM
    i. For each key that has a valid preserved value in the TPM
        (1) if parentPCRStatus is TRUE then call TPM_FlushSpecific(keyHandle)
        (2) if IsVolatile is TRUE then call TPM_FlushSpecifid(keyHandle)

3. If stType = TPM_ST_STATE
    a. If the TPM has no state to restore the TPM MUST set the internal state such that it returns TPM_FAILED_SELFTEST to all subsequent commands
    b. The TPM MAY determine for each session type (authorization, transport…) to release or maintain the session information. The TPM reports how it manages sessions in the GetCapability command.
    c. The TPM SHALL take all necessary actions to ensure that all PCRs contain valid preserved values. If the TPM is unable to successfully complete these actions, it SHALL enter the TPM failure mode.
    d. The TPM MAY initialize auditDigest to NULL
        i. Otherwise, the TPM SHALL take all actions necessary to ensure that auditDigest contains a valid value. If the TPM is unable to successfully complete these actions, the TPM SHALL initialize auditDigest to NULL and SHALL set the internal set such that the TPM returns TPM_FAILED_SELFTEST to all subsequent commands.
    e. The TPM MUST restore the following flags to their preserved states:
        i. TPM_STCLEAR_FLAGS -> PhysicalPresence
        ii. TPM_STCLEAR_FLAGS -> PhysicalPresenceLock
        iii. TPM_STCLEAR_FLAGS -> deactivated

iv. TPM_STCLEAR_FLAGS -> disableForceClear

f. The TPM MUST restore all keys that have been saved

g. The TPM resumes normal operation. If the TPM is unable to resume normal operation, it SHALL enter the TPM failure mode.

4. If stType = TPM_ST_DEACTIVATED

   a. Invalidate sessions

      i. Ensure that all resources associated with saved and active sessions are invalidated

   b. The TPM MUST set TPM_STCLEAR_FLAGS -> deactivated to TRUE

   c. The TPM MUST invalidate any explicitly preserved state

      i. The TPM MUST ensure that state associated with TPM_SaveState is invalidated

5. The TPM MUST set TPM_STANY_FLAGS -> postInitialise to FALSE

# 3.3 TPM_SaveState

***Start of informative comment:***

This warns a TPM to save some state information.

If the relevant shielded storage is non-volatile, this command need have no effect.

If the relevant shielded storage is volatile and the TPM alone is unable to detect the loss of external power in time to move data to non-volatile memory, this command should be presented before the TPM enters a low or no power state.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SaveState. |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Description**

1. Preserved values MUST be non-volatile.
2. If data is never stored in a volatile medium, that data MAY be used as preserved data. In such cases, no explicit action may be required to preserve that data.
3. If an explicit action is required to preserve data, it MUST be possible for the TPM to determine whether preserved data is valid.
4. If the parameter mirrored by a preserved value is altered, the preserved value MUST be declared invalid. If the parameter mirrored by any preserved value is altered, all preserved values MAY be declared invalid.

**Actions**

1. Store PCR contents except for
   a. If the PCR attribute pcrReset is TRUE
   b. PCR 15 (debug PCR)
2. The auditDigest MUST be handled according to the audit requirements as reported by TPM_GetCapability
3. Handle TPM_STANY_DATA -> currentTicks according to the values returned by tickType
4. All values in TPM_STCLEAR_DATA MUST be preserved

5. All values in TPM_STCLEAR_FLAGS MUST be preserved
6. The contents of any key that is currently loaded SHOULD be preserved if the key's parentPCRStatus indicator is FALSE and its IsVolatile indicator is FALSE.
7. The contents of any key that has TPM_KEY_CONTROL_OWNER_EVICT set MUST be preserved
8. The contents of any key that is currently loaded MAY be preserved as reported by TPM_GetCapability
9. The contents of sessions (authorization, transport etc.) MAY be preserved as reported by TPM_GetCapability

# 4. Admin Testing

## 4.1 TPM_SelfTestFull

***Start of informative comment:***

SelfTestFull tests all of the TPM capabilities.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SelfTestFull |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Actions**

1. TPM_SelfTestFull SHALL cause a TPM to perform self-test of each TPM internal function.
2. Failure of any test results in overall failure, and the TPM goes into failure mode.

# 4.2 TPM_ContinueSelfTest

***Start of informative comment:***

CotinueSelfTest informs the TPM that it may complete the self test of all TPM functions.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_ContinueSelfTest |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Actions**

1. If TPM_PERMANENT_FLAGS -> FIPS is TRUE
   a. The TPM MUST run ALL self-tests
2. Else
   a. The TPM MUST complete all self-tests that are outstanding
      i. Instead of completing all outstanding self-tests the TPM MAY run all self-tests
3. The TPM SHALL immediately respond to the caller with a return code.
   a. When TPM_ContinueSelfTest finishes execution, it SHALL NOT respond to the caller with a return code.
4. The TPM SHALL unilaterally execute the functions of TPM_ContinueSelfTest upon receipt of a command that calls a capability-X that uses untested TPM functions. If the self-test fails, the TPM SHALL return the error code TPM_FAILEDSELFTEST. If the self-test passes, the TPM SHALL execute capability-X.

# 4.3 TPM_GetTestResult

***Start of informative comment:***

TPM_GetTestResult provides manufacturer specific information regarding the results of the self test. This command will work when the TPM is in self test failure mode. The reason for allowing this command to operate in the failure mode is to allow TPM manufacturers to obtain diagnostic information.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_GetTestResult |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 4 | | | UINT32 | outDataSize | The size of the outData area |
| 5 | <> | | | BYTE[] | outData | The outData this is manufacturer specific |

**Actions**

1. The TPM SHALL respond to this command with a manufacturer specific block of information that describes the result of the latest self test.
2. The information MUST NOT contain any data that uniquely identifies an individual TPM.

# 5. Admin Opt-in

## 5.1 TPM_SetOwnerInstall

***Start of informative comment:***

When enabled but without an owner this command sets the persistent flag that allows or disallows the ability to insert an owner.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_SetOwnerInstall |
| 4 | 1 | | | BOOL | state | State to which ownership flag is to be set. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Action**

1. If the TPM has a current owner, this command immediately returns with TPM_SUCCESS.
2. The TPM validates the assertion of physical access. The TPM then sets the value of TPM_PERMANENT_FLAGS -> ownership to the value in state.

## 5.2 TPM_OwnerSetDisable

***Start of informative comment:***

The TPM owner sets the persistent disable flag

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_OwnerSetDisable |
| 4 | 1 | 2S | 1 | BOOL | disableState | Value for disable state – enable if TRUE |
| 5 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 6 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 8 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_OwnerSetDisable |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Action**

1. The TPM SHALL authenticate the command as coming from the TPM Owner. If unsuccessful, the TPM SHALL return TPM_BAD_AUTH.
2. The TPM SHALL set the TPM_PERMANENT_FLAGS -> disable flag to the value in the disableState parameter.

# 5.3 TPM_PhysicalEnable

***Start of informative comment:***

Sets the persistent disable flag to FALSE using physical presence as authorization.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_PhysicalEnablel |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Action**

1. Validate that physical presence is being asserted, if not return TPM_BAD_PRESENCE
2. The TPM SHALL set the TPM_PERMANENT_FLAGS.disable value to FALSE.

# 5.4 TPM_PhysicalDisable

***Start of informative comment:***

Sets the persistent disable flag to TRUE using physical presence as authorization

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_PhysicalDisable |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Action**

1. Validate that physical presence is being asserted, if not return TPM_BAD_PRESENCE
2. The TPM SHALL set the TPM_PERMANENT_FLAGS.disable value to TRUE.

# 5.5 TPM_PhysicalSetDeactivated

***Start of informative comment:***

Enables the TPM using physical presence as authorization.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_PhysicalSetDeactivated |
| 4 | 1 | | | BOOL | state | State to which deactivated flag is to be set. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Action**

1. Validate that physical presence is being asserted, if not return TPM_BAD_PRESENCE
2. The TPM SHALL set the TPM_PERMANENT_FLAGS.deactivated flag to the value in the state parameter.

# 5.6 TPM_SetTempDeactivated

***Start of informative comment:***

This command allows the operator of the platform to deactivate the TPM until the next boot of the platform.

This command requires operator authorization. The operator can provide the authorization by either the assertion of physical presence or presenting the operation authorization value.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_SetTempDeactivated |
| 5 | 4 | 2S | 4 | TPM_AUTHHANDLE | authHandle | The authorization handle used for operator authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 6 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 8 | 20 | | | TPM_AUTHDATA | operatorAuth | HMAC key: operatorAuth |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_OwnerSetDisable |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: operatorAuth. |

**Action**

1. If tag = TPM_TAG_REQ_AUTH1_COMMAND
   a. Validate that authHandle is TPM_KH_OPERATOR, on error return TPM_INVALID_KEYHANDLE
   b. If TPM_PERMANENT_FLAGS -> operator is FALSE return TPM_NOOPERATOR
   c. Validate command and parameters using operatorAuth, on error return TPM_AUTHFAIL
2. Else
   a. If physical presence is not asserted the TPM MUST return TPM_BADPRESENCE
3. The TPM SHALL set the TPM_STCLEAR_FLAGS.deactivated flag to the value TRUE.

# 5.7 TPM_SetOperatorAuth

***Start of informative comment:***

This command allows the setting of the operator authorization value.

There is no confidentiality applied to the operator authorization as the value is sent under the assumption of being local to the platform. If there is a concern regarding the path between the TPM and the keyboard then unless the keyboard is using encryption and a secure channel an attacker can read the values.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_SetOperatorAuth |
| 4 | 20 | | | TPM_SECRET | operatorAuth | The operator authorization |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Action**

1. If physical presence is not asserted the TPM MUST return TPM_BADPRESENCE
2. The TPM SHALL set the TPM_PERSISTENT_DATA -> operatorAuth
3. The TPM SHALL set TPM_PERMANENT_FLAGS -> operator to TRUE

# 6. Admin Ownership

## 6.1 TPM_TakeOwnership

***Start of informative comment:***

This command inserts the TPM Ownership value into the TPM.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_TakeOwnership |
| 4 | 2 | 2S | 2 | TPM_PROTOCOL_ID | protocolID | The ownership protocol in use. |
| 5 | 4 | 3S | 4 | UINT32 | encOwnerAuthSize | The size of the encOwnerAuth field |
| 6 | <> | 4S | <> | BYTE[ ] | encOwnerAuth | The owner authorization data encrypted with PUBEK |
| 7 | 4 | 5S | 4 | UINT32 | encSrkAuthSize | The size of the encSrkAuth field |
| 8 | <> | 6S | <> | BYTE[ ] | encSrkAuth | The SRK authorization data encrypted with PUBEK |
| 9 | <> | 7S | <> | TPM_KEY | srkParams | Structure containing all parameters of new SRK. pubKey.keyLength & encSize are both 0 |
| 10 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for this command |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 11 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 12 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 13 | 20 | | | TPM_AUTHDATA | ownerAuth | Authorization digest for input params. HMAC key: the new ownerAuth value. See actions for validation operations |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_TakeOwnership |
| 4 | <> | 3S | <> | TPM_KEY | srkPub | Structure containing all parameters of new SRK. srkPub.encData is set to 0. |
| 5 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 7 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: the new ownerAuth value |

**Actions**

1. If TPM_PERMANENT_DATA -> ownerAuth is valid return TPM_OWNER_SET
2. If TPM_PERMANENT_FLAGS -> ownership is FALSE return TPM_INSTALL_DISABLED
3. If TPM_PERMANENT_DATA -> endorsementKey is invalid return TPM_NO_ENDORSEMENT
4. Verify that authHandle is of type OIAP on error return TPM_AUTHFAIL
5. Create A1 a TPM_SECRET by decrypting EncOwnerAuth using PRIVEK as the key
   a. This requires that A1 was encrypted using the PUBEK
   b. Validate that A1 is a length of 20 bytes, on error return TPM_BAD_KEY_PROPERTY
6. Validate the command and parameters using A1 and ownerAuth, on error return TPM_AUTHFAIL
7. Validate srkParams
   a. If srkParams -> keyUsage is not TPM_KEY_STORAGE return TPM_INVALID_KEYUSAGE
   b. If srkParams -> migratable is TRUE return TPM_INVALID_KEYUSAGE
   c. If srkParams -> algorithmParms -> algorithmID is NOT TPM_ALG_RSA return TPM_BAD_KEY_PROPERTY
   d. If srkParams -> algorithmParms -> encScheme is NOT TPM_ES_RSAESOAEP_SHA1_MGF1 return TPM_BAD_KEY_PROPERTY
   e. If srkParams -> algorithmParms -> sigScheme is NOT TPM_SS_NONE return TPM_BAD_KEY_PROPERTY
   f. If srkParams -> algorithmParms -> parms -> keyLength MUST be greater than or equal to 2048, on error return TPM_BAD_KEY_PROPERTY
8. Generate K1 according to the srkParams on error return TPM_BAD_KEY_PROPERTY
9. Create A2 a TPM_SECRET by decrypting EncSrkAuth using the PRIVEK
   a. This requires A2 to be encrypted using the PUBEK
   b. Validate that A1 is a length of 20 bytes, on error return TPM_BAD_KEY_PROPERTY
   c. Store A2 in K1 -> authData
10. Store K1 in TPM_PERMANENT_DATA -> srk
11. Store A1 in TPM_PERMANENT_DATA -> ownerAuth
12. Create TPM_PERMANENT_DATA -> contextKey according to the rules for the algorithm in use by the TPM to save context blobs
13. Create TPM_PERMANENT_DATA -> delegateKey according to the rules for the algorithm in use by the TPM to save delegate blobs
14. Create TPM_PERMANENT_DATA -> tpmProof by using the TPM RNG
15. Export TPM_PERMANENT_DATA -> srk as srkPub
16. Calculate resAuth using the newly established TPM_PERMANENT_DATA -> ownerAuth

# 6.2 TPM_OwnerClear

***Start of informative comment:***

The OwnerClear command performs the clear operation under Owner authorization. This command is available until the Owner executes the DisableOwnerClear, at which time any further invocation of this command returns TPM_CLEAR_DISABLED.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_OwnerClear |
| 4 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 5 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Ignored |
| 7 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_OwnerClear |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Fixed value FALSE |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: old ownerAuth. |

**Actions**

1. Verify that the TPM Owner authorizes the command and all of the input, on error return TPM_AUTHFAIL.
2. If TPM_PERMANENT_FLAGS -> DisableOwnerClear is TRUE then return TPM_CLEAR_DISABLED.
3. Unload all loaded keys.
4. The TPM MUST NOT modify the following TPM_PERMANENT_DATA items
   a. endorsementKey
   b. revMajor

c. revMinor
d. tickType
e. manuMaintPub
f. auditMonotonicCounter
g. monotonicCounter
h. pcrAttrib
i. rngState
j. fipsReset
k. maxNVBufSize

5. The TPM MUST invalidate the following TPM_PERMANENT_DATA items and any internal resources associated with these items
   a. ownerAuth
   b. srk
   c. delegateKey
   d. contextKey
   e. tpmProof
   f. operatorAuth
6. The TPM MUST reset to manufacturing defaults the following TPM_PERMANENT_DATA items
   a. None
7. The TPM MUST invalidate all fields of TPM_STANY_DATA
8. The TPM MUST invalidate all fields of TPM_STCLEAR_DATA
9. The TPM MUST set the following TPM_PERMANENT_FLAGS to their default values
   a. disable
   b. deactivated
   c. readPubek
   d. disableOwnerClear
10. The TPM MUST set TPM_PERMANENT_FLAGS -> ownership to FALSE
11. The TPM MUST set TPM_PERMANENT_FLAGS -> operator to FALSE
12. The TPM releases all TPM_PERMANENT_DATA -> monotonicCounter settings
    a. This includes invalidating all currently allocated counters. The result will be no currently allocated counters and the new owner will need to allocate counters. The actual count value will continue to increase.
13. The TPM MUST NOT deallocate any currently defined NV storage areas
14. The TPM MUST invalidate all familyTable entries

## 6.3 TPM_ForceClear

***Start of informative comment:***

The ForceClear command performs the Clear operation under physical access. This command is available until the execution of the DisableForceClear, at which time any further invocation of this command returns TPM_CLEAR_DISABLED.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_ForceClear |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Actions**

1. The TPM SHALL check for the assertion of physical presence, if not present return TPM_BAD_PRESENCE
2. If TPM_STCLEAR_FLAGS -> disableForceClear is TRUE return TPM_CLEAR_DISABLED
3. The TPM SHALL execute the actions of TPM_OwnerClear (except for the TPM Owner authorization check)

# 6.4 TPM_DisableOwnerClear

***Start of informative comment:***

The DisableOwnerClear command disables the ability to execute the TPM_OwnerClear command permanently. Once invoked the only method of clearing the TPM will require physical access to the TPM.

After the execution of TPM_ForceClear, ownerClear is re-enabled and must be explicitly disabled again by the new TPM Owner.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_DisableOwnerClear |
| 4 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 5 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 7 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_DisableOwnerClear |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Actions**

1. The TPM verifies that the authHandle properly authorizes the owner.
2. The TPM sets the TPM_PERMANENT_FLAGS -> disableownerclear flag to TRUE.
3. When this flag is TRUE the only mechanism that can clear the TPM is the TPM_ForceClear command. The TPM_ForceClear command requires physical access to the TPM to execute.

# 6.5 TPM_DisableForceClear

Start of informative comment:

The DisableForceClear command disables the execution of the ForceClear command until the next startup cycle. Once this command is executed, the TPM_ForceClear is disabled until another startup cycle is run.

End of informative comment.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_DisableForceClear |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Actions**

1. The TPM sets the TPM_STCLEAR_FLAGS.disableforceclear flag in the TPM that disables the execution of the TPM_ForceClear command.

# 6.6 TSC_PhysicalPresence

***Start of informative comment:***

Some TPM operations require the indication of a human's physical presence at the platform. The presence of the human either provides another indication of platform ownership or a mechanism to ensure that the execution of the command is not the result of a remote software process.

This command allows a process on the platform to indicate the assertion of physical presence. As this command is executable by software there must be protections against the improper invocation of this command.

The physicalPresenceHWEnable and physicalPresenceCMDEnable indicate the ability for either SW or HW to indicate physical presence. These flags can be reset until the physicalPresenceLifetimeLock is set. The platform manufacturer should set these flags to indicate the capabilities of the platform the TPM is bound to.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TSC_ORD_PhysicalPresence. |
| 4 | 2 | | | TPM_PHYSICAL_PRESENCE | physicalPresence | The state to set the TPM's Physical Presence flags. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. See section 4.3 of Main Specification. |

**Descriptions**

**Actions**

1. If TPM_PERMANENT_FLAGS -> physicalPresenceLifetimeLock is FALSE
   a. If physicalPresence -> TPM_PHYSICAL_PRESENCE_HW_ENABLE is TRUE
      i. Set TPM_PERMANENT_FLAGS -> physicalPresenceHWEnable to TRUE
   b. If physicalPresence -> TPM_PHYSICAL_PRESENCE_CMD_ENABLE is TRUE
      i. Set TPM_PERMANENT_FLAGS -> physicalPresenceCMDEnable to TRUE
2. If physicalPresence -> TPM_PHYSICAL_PRESENCE_LIFETIME_LOCK is TRUE
   a. Set TPM_PERMANENT_FLAGS -> physicalPresenceLifetimeLock to TRUE
3. If TPM_PERMANENT_FLAGS -> physicalPresenceCMDEnable is TRUE and TPM_STCLEAR_FLAGS -> physicalPresenceLock is FALSE.
   a. If physicalPresence -> TPM_PHYSICAL_PRESENCE_PRESENT is TRUE

i. Set TPM_STCLEAR_FLAGS -> physicalPresence to TRUE

b. If physicalPresence -> TPM_PHYSICAL_PRESENCE_NOTPRESENT is TRUE

i. Set TPM_STCLEAR_FLAGS -> physicalPresence to FALSE

c. if physicalPresence -> TPM_PHYSICAL_PRESENCE_LOCK is TRUE

i. Set TPM_STCLEAR_FLAGS -> physicalPresenceLock to TRUE

4. Else

a. Return TPM_BAD_PARAMETER

# 6.7 TSC_ResetEstablishmentBit

***Start of informative comment:***

The PC TPM Interface Specification (TIS) specifies a bit that is set upon execution of the HASH_START sequence. The setting of this implies the creation of a Trusted Operating System on the platform.

There are reasons to reset the bit. This command allows for the resetting of the bit under controlled circumstances.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TSC_ORD_ResetEstablishmentBit |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Actions**

1. Validate the assertion of locality 3 or locality 4
2. Set the tpmEstablished bit contained in the TPM_ACCESS register to 1
3. Return TPM_SUCCESS

# 7. The GetCapability Commands

***Start of informative comment:***

The TPM has numerous capabilities that a remote entity may wish to know about. These items include support of algorithms, key sizes, protocols and vendor-specific additions. The GetCapability command allows the TPM to report back to the requestor what type of TPM it is dealing with.

The request for information requires the requestor to specify which piece of information that is required. The request does not allow the "merging" of multiple requests and returns only a single piece of information.

In failure mode, the TPM returns a limited set of information that includes the TPM manufacture and model and the SelfTestResult

In version 1.2 with the deletion of GetCapabilitySigned the way to obtain a signed listing of the capabilities is to create a transport session, perform GetCapability commands to list the information and then close the transport session using ReleaseTransportSigned.

***End of informative comment.***

1. The standard information provided in TPM_GetCapability MUST NOT provide unique information
   a. The TPM has no control of information placed into areas on the TPM like the NV store that is reported by the TPM. Configuration information for these areas could conceivably be unique

# 7.1 TPM_GetCapability

***Start of informative comment:***

This command returns current information regarding the TPM.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_GetCapability |
| 4 | 4 | | | TPM_CAPABILITY_AREA | capArea | Partition of capabilities to be interrogated |
| 5 | 4 | | | UINT32 | subCapSize | Size of subCap parameter |
| 6 | <> | | | BYTE[] | subCap | Further definition of information |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 4 | | | UINT32 | respSize | The length of the returned capability response |
| 5 | <> | | | BYTE[ ] | resp | The capability response |

**Actions**

The TPM validates the capArea and subCap indicators. If the information is available, the TPM creates the response field and fills in the actual information.

The num column is merely for help in labeling the table and is informative only.

**TPM response to various requests**

| Num | CapArea | subCap | Response |
|---|---|---|---|
| 1 | TPM_CAP_ALG | TPM_ALG_XX:<br>A value of TPM_ALGORITHM_ID | Boolean value. TRUE means that the TPM supports the algorithm for TPM_Sign, TPM_Seal, TPM_UnSeal and TPM_Unbind and related commands. FALSE indicates that for these types of commands the algorithm is not supported. |
| 2 | TPM_CAP_KEY_STATUS | handle | Boolean value of ownerEvict. The handle MUST point to a valid key handle. |
| 3 | TPM_CAP_PROPERTY | TPM_CAP_PROP_DAA_MAX | UINT32. The maximum number of DAA sessions (join or sign) that the TPM supports |
| 4 | TPM_CAP_CHECK_LOADED | ALGORITHM:<br>A value of TPM_KEY_PARMS | A Boolean value. TRUE indicates that the TPM has enough memory available to load a key of the type specified by ALGORITHM. FALSE indicates that the TPM does not have enough memory. |

| | | | |
|---|---|---|---|
| 5 | TPM_CAP_FLAG | TPM_CAP_FLAG_PERSISTENT | The TPM_PERMANENT_FLAGS structure |
| 6 | TPM_CAP_FLAG | TPM_CAP_FLAG_VOLATILE | The TPM_STCLEAR_FLAGS structure |
| 7 | TPM_CAP_PROPERTY | TPM_CAP_PROP_DELEGATE_ROW | UINT32. The size of the delegate table in rows. |
| 8 | TPM_CAP_ORD | ORDINAL:<br>A value of command ordinal: | Boolean value. TRUE indicates that the TPM supports the ordinal. FALSE indicates that the TPM does not support the ordinal. |
| 9 | TPM_CAP_PID | TPM_PID:<br>A value of TPM_PROTOCOL_ID: | Boolean value. TRUE indicates that the TPM supports the protocol, FALSE indicates that the TPM does not support the protocol. |
| 10 | TPM_CAP_PROPERTY | TPM_CAP_PROP_PCR | UINT32 value. Returns the number of PCR registers supported by the TPM |
| 11 | TPM_CAP_PROPERTY | TPM_CAP_PROP_DIR_AUTH | UINT32 value. Returns the number of DIR registers under control of the TPM owner supported by the TPM. |
| 12 | TPM_CAP_PROPERTY | TPM_CAP_PROP_MANUFACTURER | UINT32 value. Returns the Identifier of the TPM manufacturer. |
| 13 | TPM_CAP_PROPERTY | TPM_CAP_PROP_KEYS | UINT32 value. Returns the number of 2048-bit RSA keys that can be loaded. This MAY vary with time and circumstances. |
| 14 | TPM_CAP_PROPERTY | TPM_CAP_PROP_TIS | These report the platform specific communication timeout values. For further detail refer to the platform specific specification |
| 15 | TPM_CAP_PROPERTY | TPM_CAP_MIN_COUNTER | UINT32. The minimum amount of time in 10ths of a second that must pass between invocations of incrementing the monotonic counter. |
| 16 | TPM_CAP_PROPERTY | TPM_CAP_PROP_SESSIONS | UNIT32. The number of available sessions from the pool. This MAY vary with time and circumstances. Pool sessions include authorization and transport sessions. |
| 17 | TPM_CAP_PROPERTY | TPM_CAP_PROP_CONTEXT_DIST | UINT32. The maximum distance between context count values. This MUST be at least 2^16. |
| 18 | TPM_CAP_PROPERTY | TPM_CAP_PROP_COUNTERS | UINT32. The number of available monotonic counters. This MAY vary with time and circumstances. |
| 19 | TPM_CAP_VERSION | Ignored | TPM_Version structure. The version MUST be 1.2.0.0 |
| 20 | TPM_CAP_PROPERTY | TPM_CAP_PROP_MAX_SESSIONS | UINT32. The maximum number of available sessions. Sessions include authorization and transport sessions. |
| 21 | TPM_CAP_MANUFACTURER_VER | Ignored | TPM_Version structure. The version MUST be 1.2.x.x where the x's represent the manufacturer version of the TPM |
| 22 | TPM_CAP_PROPERTY | TPM_CAP_PROP_MAX_COUNTERS | UINT32. The maximum number of monotonic counters under control of TPM_CreateCounter |
| 23 | TPM_CAP_PROPERTY | TPM_CAP_PROP_MAX_KEYS | UINT32. The maximum number of 2048 RSA keys that the TPM can support. The number does not include the EK or SRK. |
| 24 | TPM_CAP_PROPERTY | TPM_CAP_PROP_DAA_INTERRUPT | BOOL. A value of TRUE indicates that the TPM will accept ANY command while executing a DAA Join or Sign.<br>A value of FALSE indicates that the TPM will invalidate the DAA Join or Sign upon the receipt of any command other than the next join/sign in the session or a TPM_SaveContext |
| 25 | TPM_CAP_PROPERTY | TPM_CAP_PROP_OWNER | BOOL. A value of TRUE indicates that the TPM has successfully installed an owner. |
| 26 | TPM_CAP_PROPERTY | TPM_CAP_PROP_CONTEXT | UINT32. The number of available context slots. This MAY vary with time and circumstances. |
| 27 | TPM_CAP_PROPERTY | TPM_CAP_PROP_MAX_CONTEXT | UINT32. The maximum number of context slots. |
| 28 | TPM_CAP_PROPERTY | TPM_CAP_PROP_STARTUP_EFFECT | The TPM_STARTUP_EFFECTS structure |
| 30 | TPM_CAP_PROPERTY | TPM_CAP_PROP_FAMILYROWS | UINT32. The number of rows in the family table |

| | | | |
|---|---|---|---|
| 31 | TPM_CAP_HANDLE | TPM_RESOURCE_TYPE | A TPM_KEY_HANDLE_LIST structure that enumerates all handles currently loaded in the TPM for the given resource type.<br>When describing keys the handle list only contains the number of handles that an external manager can operate with and does not include the EK or SRK.<br>Legal resources are TPM_RT_KEY, TPM_RT_AUTH, TPM_RT_TRANS, TPM_RT_DELEGATE |
| 32 | TPM_CAP_PROP_PROPERTY | TPM_CAP_PROP_NV_AVAILABLE | UINT32 the maximum available NV space left in the TPM. This area is a total count of bytes available and does not indicate that a contigous area of this size can be allocated |
| 33 | TPM_CAP_PROP_PROPERTY | TPM_CAP_PROP_NV_MAXBUF | UINT32 the maixum size that a NV allocation can take. This value is reflected in TPM_PERMANENT_DATA -> maxNVBufSize |
| 34 | TPM_CAP_NV_INDEX | TPM_NV_INDEX | A TPM_NV_DATA_PUBLIC structure that indicates the values for the TPM_NV_INDEX |
| 35 | TPM_CAP_NV_LIST | ignored | A list of UINT32 that are the NV storage indexes. |
| 36 | TPM_CAP_TRANS_ALG | TPM_ALG_XXX | Boolean value. TRUE means that the TPM supports the algorithm for TPM_EstablishTransport, TPM_ExecuteTransport and TPM_ReleaseTransportSigned. FALSE indicates that for these three commands the algorithm is not supported." |
| 37 | TPM_CAP_FLAG | TPM_CAP_FLAG_STANY | The TPM_STANY_FLAGS structure |
| 38 | TPM_CAP_GPIO_CHA NNEL | UINT16 TPM_PLATFORM_SPECIFIC<br>UINT16 channel number | Boolean value – TRUE the TPM supports the channel and FALSE the TPM does not support the channel |
| 39 | TPM_CAP_PROP_PROPERTY | TPM_CAP_PROP_CMK_RESTRICTION | TPM_Permanent_Data -> restrictDelegation |
| 40 | TPM_CAP_KEY_HANDLE | Ignored | A TPM_KEY_HANDLE_LIST structure that enumerates all key handles loaded on the TPM. The list only contains the number of handles that an external manager can operate with and does not include the EK or SRK.<br>This is command is available for backwards compability. It is the same as item 31 with a resouce type of keys. |
| 41 | TPM_CAP_TRANS_ES | TPM_EX_XXX | Boolean value. TRUE means the TPM supports the encryption scheme in a transport session. |

# 8. Auditing

## 8.1 Audit Generation

***Start of informative comment:***

The TPM generates an audit event in response to the TPM executing a function that has the audit flag set to TRUE for that function.

The TPM maintains an extended value for all audited operations.

Input audit generation occurs before the listed actions and output audit generation occurs after the listed actions.

***End of informative comment.***

**Description**

The TPM extends the audit digest whenever the ordinalAuditStatus is TRUE for the ordinal about to be executed.

**Actions**

The TPM will execute the ordinal and perform auditing in the following manner

1. Map V1 to TPM_STANY_DATA
2. Map P1 to TPM_PERMANENT_DATA
3. If V1 -> auditDigest is NULL
   a. Increment P1 -> auditMonotonicCounter by 1
4. Create A1 a TPM_AUDIT_EVENT_IN structure
   a. Set A1 -> inputParms to the input parameters from the command
   b. Set A1 -> ordinal to the ordinal of the command
   c. Set A1 -> auditCount to P1 -> auditMonotonicCounter
   d. Set V1 -> auditDigest to SHA-1 (V1 -> auditDigest || A1)
5. Execute command
   a. Execution implies the performance of the listed actions for the ordinal.
6. Create A2 a TPM_AUDIT_EVENT_OUT structure
   a. Set A2 -> outputParms to the output parameters from the command
   b. Set A2 -> returnCode to the return code for the command
   c. Set A2 -> ordinal to the ordinal of the command
   d. Set A2 -> auditCount to P1 -> auditMonotonicCounter
   e. Set V1 -> auditDigest to SHA-1 (V1 -> auditDigest || A2)

## 8.2 Effect of audit failing after successful completion of a command

***Start of informative comment:***

An operation could complete successfully and then when the TPM attempts to audit the command the audit process could have an internal error that forces the TPM to return an error.

The TPM is unable to return the results of the command that ran and this includes success or failure. To indicate to the caller the TPM will one of two error codes TPM_AUDITFAIL_SUCCESSFUL and TPM_AUDITFAIL_UNSUCCESSFUL. These two error codes indicate if the command succeeded or failed. The purpose of this command is to indicate to the caller what occurred with the command execution.

This is new functionality that changes the 1.1 TPM functionality when this condition occurs.

***End of informative comment.***

1. When after successful completion of an operation, and in performing the audit process, the TPM has an internal failure (unable to write, SHA-1 failure etc.) the TPM MUST set the internal TPM state such that the TPM returns the TPM_FAILEDSELFTEST error.
2. If the command is returning a return code that indicates successful execution of the command the TPM SHALL change the return code to TPM_AUDITFAIL_SUCCESSFUL. For all other error codes the TPM MUST return TPM_AUDITFAIL_UNSUCCESSFUL.
3. If the TPM is permanently nonrecoverable after an audit failure, then the TPM MUST always return TPM_FAILEDSELFTEST for every command other than TPM_GetTestResult. This state must persist regardless of power cycling, the execution of TPM_Init or any other actions.

# 8.3 TPM_GetAuditDigest

***Start of informative comment:***

This returns the current audit digest. The external audit log has the responsibility to track the parameters that constitute the audit digest.

This value may be unique to an individual TPM. The value however will be changing at a rate set by the TPM Owner. Those attempting to use this value may find it changing without their knowledge. This value represents a very poor source of tracking uniqueness.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | Ordinal | Command ordinal: TPM_ORD_GetAuditDigest |
| 4 | 4 | | | UINT32 | startOrdinal | The starting ordinal for the list of audited ordinals |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | Tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 5 | 10 | | | TPM_COUNTER_VALUE | counterValue | The current value of the audit monotonic counter |
| 4 | 20 | | | TPM_DIGEST | auditDigest | Log of all audited events |
| 5 | 1 | | | BOOL | more | TRUE if the output does not contain a full list of audited ordinals |
| 5 | 4 | | | UINT32 | ordSize | Size of the ordinal list in bytes |
| 6 | <> | | | UINT32[] | ordList | List of ordinals that are audited. |

**Actions**

1. The TPM sets auditDigest to TPM_STANY_DATA -> auditDigest
2. The TPM sets counterValue to TPM_PERMANENT_DATA -> auditMonotonicCounter
3. The TPM creates an ordered list of audited ordinals. The list starts at startOrdinal listing each ordinal that is audited.
   a. If startOrdinal is 0 then the first ordinal that could be audited would be TPM_OIAP (ordinal 0x0000000A)
   b. The next ordinal would be TPM_OSAP (ordinal 0x0000000B)
4. If the ordered list does not fit in the output buffer the TPM sets more to TRUE

# 8.4 TPM_GetAuditDigestSigned

***Start of informative comment:***

The signing of the audit log returns the entire digest value and the list of currently audited commands.

The inclusion of the list of audited commands as an atomic operation is to tie the current digest value with the list of commands that are being audited.

The signing functionality of this command could be handled by a signed transport session. The resetting of the audit log functionality must remain in this command; hence there is no way to remove this ordinal from the set of active ordinals a TPM must support.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | ordinal: TPM_ORD_GetAuditDigestSigned |
| 4 | 4 | | | TPM_KEY_HANDLE | keyHandle | The handle of a loaded key that can perform digital signatures. |
| 5 | 4 | 2S | 4 | UINT32 | startOrdinal | The starting ordinal for the ordered list |
| 6 | 1 | 3S | 1 | BOOL | closeAudit | Indication if audit session should be closed |
| 7 | 20 | 4S | 20 | TPM_NONCE | antiReplay | A nonce to prevent replay attacks |
| 8 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for key authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 9 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 10 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 11 | 20 | | | TPM_AUTHDATA | keyAuth | Authorization. HMAC key: key.usageAuth. |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | ordinal: TPM_ORD_GetAuditDigestSigned |
| 4 | 10 | 3S | 10 | TPM_COUNTER_VALUE | counterValue | The value of the audit monotonic counter |
| 5 | 20 | 4S | 20 | TPM_DIGEST | auditDigest | Log of all audited events |
| 6 | 1 | 5S | 1 | BOOL | more | TRUE if the output does not contain a full list of audited ordinals |
| 7 | 4 | 5S | 4 | UINT32 | ordSize | The size of the ordinal list in bytes |
| 8 | <> | 6S | <> | UINT32[] | ordinalList | The list of ordinals that are being audited |
| 9 | 4 | 7S | 4 | UINT32 | sigSize | The size of the sig parameter |
| 10 | <> | 8S | <> | BYTE[] | sig | The signature of the area |
| 11 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 12 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 13 | 20 | | | TPM_AUTHDATA | resAuth | Authorization HMAC key: key.usageAuth. |

**Actions**

1. Call TPM_GetAuditDigest passing startOrdinal and obtain auditDigest, counterValue, ordSize and ordinalList
   a. The TPM MUST ensure that ordinal list AND the resulting signature will fit in the output buffer
   b. The TPM MAY truncate the ordinal list returned by TPM_GetAuditDigest
   c. If GetAuditDigest truncates the ordinalList or this command truncates the list the TPM MUST set more to TRUE
2. Create D1 a TPM_SIGN_INFO structure and set the structure defaults
   a. Set D1 -> fixed to “ADIG”
   b. Set D1 -> replay to antiReplay
   c. Create D2 the concatenation of auditDigest || counterValue || ordinalList
   d. Set D1 -> dataLen to the length of D2
   e. Set D1 -> data to D2
   f. Create a digital signature of D1 by using the signature scheme for keyHandle
3. If closeAudit == TRUE and keyHandle->keyUsage is TPM_KEY_IDENTITY
   a. TPM_STANY_DATA -> auditDigest MUST be set to NULLS.
4. Else
   a. TPM_INVALID_KEYUSAGE
5. Return the signature in the sig parameter

# 8.5 TPM_SetOrdinalAuditStatus

***Start of informative comment:***

Set the audit flag for a given ordinal. This command requires the authorization of the TPM Owner.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_SetOrdinalAuditStatus |
| 4 | 4 | 2S | 4 | TPM_COMMAND_CODE | ordinalToAudit | The ordinal whose audit flag is to be set |
| 5 | 1 | 3S | 1 | BOOL | auditState | Value for audit flag |
| 6 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 7 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 9 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_SetOrdinalAuditStatus |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Actions**

1. Validate the authorization to execute the command and the parameters
2. Validate that the ordinal points to a valid TPM ordinal, return TPM_BAD_INDEX on error
3. Set the non-volatile flag associated with ordinalToAudit to the value in auditState

# 9. Administrative Functions - Management

## 9.1 TPM_FieldUpgrade

***Start of informative comment:***

The TPM needs a mechanism to allow for updating the protected capabilities once a TPM is in the field. Given the varied nature of TPM implementations there will be numerous methods of performing an upgrade of the protected capabilities. This command, when implemented, provides a manufacturer specific method of performing the upgrade.

The manufacturer can determine, within the listed requirements, how to implement this command. The command may be more than one command and actually a series of commands.

The IDL definition is to create an ordinal for the command, however the remaining parameters are manufacturer specific.

***End of informative comment.***

**IDL Definition**

```
TPM_RESULT TPM_FieldUpgrade(
   [in, out] TPM_AUTH* ownerAuth,
   …);
```

**Type**

This is an optional command and a TPM is not required to implement this command in any form.

**Parameters**

| Type | Name | Description |
|---|---|---|
| TPM_AUTH | ownerAuth | Authentication from TPM owner to execute command |
| … | | Remaining parameters are manufacturer specific |

**Descriptions**

The upgrade mechanisms in the TPM MUST not require the TPM to hold a global secret. The definition of global secret is a secret value shared by more than one TPM.

The TPME is not allowed to pre-store or use unique identifiers in the TPM for the purpose of field upgrade. The TPM MUST NOT use the endorsement key for identification or encryption in the upgrade process. The upgrade process MAY use a TPM Identity to deliver upgrade information to specific TPM's.

The upgrade process can only change protected capabilities.

The upgrade process can only access data in shielded locations where this data is necessary to validate the TPM Owner, validate the TPME and manipulate the blob

The TPM MUST be conformant to the TPM specification, protection profiles and security targets after the upgrade. The upgrade MAY NOT decrease the security values from the original security target.

The security target used to evaluate this TPM MUST include this command in the TOE.

**Actions**

The TPM SHALL perform the following when executing the command:

1. Validate the TPM Owners authorization to execute the command

2. Validate that the upgrade information was sent by the TPME. The validation mechanism MUST use a strength of function that is at least the same strength of function as a digital signature performed using a 2048 bit RSA key.
3. Validate that the upgrade target is the appropriate TPM model and version.
4. Process the upgrade information and update the protected capabilities
5. Set the TPM_PERMANENT_DATA.revMajor and TPM_PERMANENT_DATA.revMinor to the values indicated in the upgrade. The selection of the value is a manufacturer option. The values MUST be monotonically increasing. Installing an upgrade with a major and minor revision that is less than currently installed in the TPM is a valid operation.
6. Set the TPM_STCLEAR_FLAGS.deactivated to TRUE.

# 9.2 TPM_SetRedirection

***Informative comment***

The redirection command attaches a key to a redirection receiver.

When making the connection to a GPIO channel the authorization restrictions are set at connection time and not for each invocation that uses the channel.

***End of informative comments***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SetRedirection |
| 4 | 4 | | | TPM_KEY_HANDLE | keyHandle | The keyHandle identifier of a loaded key that can implement redirection. |
| 5 | 4 | 2S | 4 | TPM_REDIR_COMMAND | redirCmd | The command to execute |
| 6 | 4 | 3S | 4 | UINT32 | inputDataSize | The size of the input data |
| 7 | <> | 4S | <> | BYTE | inputData | Manufacturer parameter |
| 8 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 9 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 10 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 11 | 20 | | | TPM_AUTHDATA | ownerAuth | HMAC key TPM Owner authorization |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_SetRedirection |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth |

**Action**

1. If tag == TPM_TAG_REQ_AUTH1_COMMAND

a. Validate the command and parameters using TPM Owner authorization, on error return TPM_AUTHFAIL

2. if redirCmd == TPM_REDIR_GPIO
   a. Validate that keyHandle points to a loaded key, return TPM_INVALID_KEYHANDLE on error
   b. Validate the key attributes specify redirection, return TPM_BAD_TYPE on error
   c. Validate that inputDataSize is 4, return TPM_BAD_SIZE on error
   d. Validate that inputData points to a valid GPIO channel, return TPM_BAD_PARAMETER on error
   e. Map C1 to the TPM_GPIO_CONFIG_CHANNEL structure indicated by inputData
   f. If C1 -> attr specifies TPM_GPIO_ATTR_OWNER
      i. If tag != TPM_TAG_REQ_AUTH1_COMMAND return TPM_AUTHFAIL
   g. If C1 -> attr specifies TPM_GPIO_ATTR_PP
      i. If TPM_STCLEAR_FLAGS -> physicalPresence == FALSE, then return TPM_BAD_PRESENCE
   h. Return TPM_SUCCESS
3. The TPM MAY support other redirection types. These types may be specified by TCG or provided by the manufacturer.

# 10. Storage functions

## 10.1 TPM_Seal

***Start of informative comment:***

The SEAL operation allows software to explicitly state the future "trusted" configuration that the platform must be in for the secret to be revealed. The SEAL operation also implicitly includes the relevant platform configuration (PCR-values) when the SEAL operation was performed. The SEAL operation uses the tpmProof value to BIND the blob to an individual TPM.

If the UNSEAL operation succeeds, proof of the platform configuration that was in effect when the SEAL operation was performed is returned to the caller, as well as the secret data. This proof may, or may not, be of interest. If the SEALed secret is used to authenticate the platform to a third party, a caller is normally unconcerned about the state of the platform when the secret was SEALed, and the proof may be of no interest. On the other hand, if the SEALed secret is used to authenticate a third party to the platform, a caller is normally concerned about the state of the platform when the secret was SEALed. Then the proof is of interest.

For example, if SEAL is used to store a secret key for a future configuration (probably to prove that the platform is a particular platform that is in a particular configuration), the only requirement is that that key can be used only when the platform is in that future configuration. Then there is no interest in the platform configuration when the secret key was SEALed. An example of this case is when SEAL is used to store a network authentication key.

On the other hand, suppose an OS contains an encrypted database of users allowed to log on to the platform. The OS uses a SEALED blob to store the encryption key for the user-database. However, the nature of SEAL is that any SW stack can SEAL a blob for any other software stack. Hence the OS can be attacked by a second OS replacing both the SEALED-blob encryption key, and the user database itself, allowing untrusted parties access to the services of the OS. To thwart such attacks, SEALED blobs include the past SW configuration. Hence, if the OS is concerned about such attacks, it may check to see whether the past configuration is one that is known to be trusted.

TPM_Seal requires the encryption of one parameter ("Secret"). For the sake of uniformity with other commands that require the encryption of more than one parameter, the string used for XOR encryption is generated by concatenating a nonce (created during the OSAP session) with the session shared secret and then hashing the result.

***End of informative comment.***

### Incoming Operands and Sizes

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Seal. |
| 4 | 4 | | | TPM_KEY_HANDLE | keyHandle | Handle of a loaded key that can perform seal operations. |
| 5 | 20 | 2S | 20 | TPM_ENCAUTH | encAuth | The encrypted authorization data for the sealed data. The encryption key is the shared secret from the OSAP protocol. |
| 6 | 4 | 3S | 4 | UINT32 | pcrInfoSize | The size of the pcrInfo parameter. If 0 there are no PCR registers in use |
| 7 | <> | 4S | <> | TPM_PCR_INFO | pcrInfo | The PCR selection information. The caller MAY use TPM_PCR_INFO_LONG. |
| 8 | 4 | 5S | 4 | UINT32 | inDataSize | The size of the inData parameter |
| 9 | <> | 6S | <> | BYTE[ ] | inData | The data to be sealed to the platform and any specified PCRs |
| 10 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization. Must be an OS_AP session for this command. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 11 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 12 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Ignored |
| 13 | 20 | | | TPM_AUTHDATA | pubAuth | The authorization digest for inputs and keyHandle. HMAC key: key.usageAuth. |

### Outgoing Operands and Sizes

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Seal. |
| 4 | <> | 3S | 4 | TPM_STORED_DATA | sealedData | Encrypted, integrity-protected data object that is the result of the TPM_Seal operation. |
| 5 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, fixed value of FALSE |
| 7 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth. |

### Descriptions

The string used for XOR encryption of the command variable named encAuth SHALL be the digest created by concatenating the shared session secret with the even numbered hash (generated by the TPM) and hashing the concatenated value.

TPM_Seal is used to encrypt private objects that can only be decrypted using TPM_Unseal.

**Actions**

1. If the inDataSize is 0 the TPM returns TPM_BAD_PARAMETER
2. If the keyUsage field of the key indicated by keyHandle does not have the value TPM_KEY_STORAGE, the TPM must return the error code TPM_INVALID_KEYUSAGE.
3. If the keyHandle points to a migratable key then the TPM MUST return the error code TPM_INVALID_KEY_USAGE.
4. Determine the version of pcrInfo
   a. If pcrInfoSize is 0
      i. set V1 to 1
   b. Else
      i. Point X1 as TPM_PCR_INFO_LONG structure to pcrInfo
      ii. If X1 -> tag is TPM_TAG_PCR_INFO_LONG
         (1) Set V1 to 2
      iii. Else
         (1) Set V1 to 1
5. If V1 is 1 then
   a. Create S1 a TPM_STORED_DATA structure
6. else
   a. Create S1 a TPM_STORED_DATA12 structure
7. Set s1 -> encDataSize to 0
8. Set s1 -> encData to NULL
9. Set s1 -> sealInfoSize to pcrInfoSize
10. If pcrInfoSize is not 0 then
    a. if V1 is 1 then
       i. Validate pcrInfo as a valid TPM_PCR_INFO structure, return TPM_BADINDEX on error
       ii. Create h1 the composite hash of the PCR selected by pcrInfo -> pcrSelection
       iii. Set s1 -> sealInfo -> digestAtCreation to h1
       iv. Set s1 -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease
    b. else
       i. Validate pcrInfo as a valid TPM_PCR_INFO_LONG structure, return TPM_BADINDEX on error
       ii. Set s1 -> creationPCRSelection to pcrInfo -> creationPCRSelection
       iii. Set s1 -> releasePCRSelection to pcrInfo -> releasePCRSelection
       iv. Set s1 -> digestAtRelease to pcrInfo -> digestAtRelease
       v. Create h2 the composite hash of the PCR selected by pcrInfo -> creationPCRSelection
       vi. Set s1 -> digestAtCreation to h2
       vii. Set s1 -> localityAtCreation to TPM_STANY_DATA -> localityModifier

11. Create X1 the SHA-1 result of the concatenation of (keyHandle -> sharedSecret || authLastNonceEven)
    a. Create a1 the XOR of encAuth and X1
    b. The TPM provides NO validation of a1. Well-known values (like NULLS) are valid and possible.
12. Create s2 a TPM_SEALED_DATA structure
    a. Set s2 -> payload to TPM_PT_SEAL
    b. Set s2 -> tpmProof to TPM_PERMANENT_DATA -> tpmProof
    c. Create h2 the SHA-1 of s1
    d. Set s2 -> storedDigest to h2
    e. Set s2 -> authData to a1
    f. Set s2 -> dataSize to inDataSize
    g. Set s2 -> data to inData
13. Validate that the size of s2 can be encrypted by the key pointed to by keyHandle, return TPM_BAD_DATASIZE on error
14. Create s3 the encryption of s2 using the key pointed to by keyHandle
15. Set continueAuthSession to FALSE
16. Set s1 -> encDataSize to the size of s3
17. Set s1 -> encData to s3
18. Return s1 as sealedData

# 10.2 TPM_Unseal

***Start of informative comment:***

The TPM_Unseal operation will reveal TPM_Sealed data only if it was encrypted on this platform and the current configuration (as defined by the named PCR contents) is the one named as qualified to decrypt it. Internally, TPM_Unseal accepts a data blob generated by a TPM_Seal operation. TPM_Unseal decrypts the structure internally, checks the integrity of the resulting data, and checks that the PCR named has the value named during TPM_Seal. Additionally, the caller must supply appropriate authorization data for blob and for the key that was used to seal that data.

If the integrity, platform configuration and authorization checks succeed, the sealed data is returned to the caller; otherwise, an error is generated.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Unseal. |
| 4 | 4 | | | TPM_KEY_HANDLE | parentHandle | Handle of a loaded key that can unseal the data. |
| 5 | <> | 2S | <> | TPM_STORED_DATA | inData | The encrypted data generated by TPM_Seal. |
| 6 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for parentHandle. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 7 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 9 | 20 | | | TPM_AUTHDATA | parentAuth | The authorization digest for inputs and parentHandle. HMAC key: parentKey.usageAuth. |
| 10 | 4 | | | TPM_AUTHHANDLE | dataAuthHandle | The authorization handle used to authorize inData. |
| | | 2H2 | 20 | TPM_NONCE | dataLastNonceEven | Even nonce previously generated by TPM |
| 11 | 20 | 3H2 | 20 | TPM_NONCE | datanonceOdd | Nonce generated by system associated with entityAuthHandle |
| 12 | 1 | 4H2 | 1 | BOOL | continueDataSession | Continue usage flag for dataAuthHandle. |
| 13 | 20 | | | TPM_AUTHDATA | dataAuth | The authorization digest for the encrypted entity. HMAC key: entity.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Unseal. |
| 4 | 4 | 3S | 4 | UINT32 | sealedDataSize | The used size of the output area for secret |
| 5 | <> | 4S | <> | BYTE[ ] | secret | Decrypted data that had been sealed |
| 6 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: parentKey.usageAuth. |
| 9 | 20 | 2H2 | 20 | TPM_NONCE | dataNonceEven | Even nonce newly generated by TPM. |
| | | 3H2 | 20 | TPM_NONCE | datanonceOdd | Nonce generated by system associated with dataAuthHandle |
| 10 | 1 | 4H2 | 1 | BOOL | continueDataSession | Continue use flag, TRUE if handle is still active |
| 11 | 20 | | | TPM_AUTHDATA | dataAuth | The authorization digest used for the dataAuth session. HMAC key: entity.usageAuth. |

**Actions**

1. The TPM MUST validate that parentAuth authorizes the use of the key in parentHandle, on error return TPM_AUTHFAIL
2. If the keyUsage field of the key indicated by parentHandle does not have the value TPM_KEY_STORAGE, the TPM MUST return the error code TPM_INVALID_KEYUSAGE.
3. The TPM MUST check that the TPM_KEY_FLAGS -> Migratable flag has the value FALSE in the key indicated by parentKeyHandle. If not, the TPM MUST return the error code TPM_INVALID_KEYUSAGE
4. Determine the version of inData
   a. If inData -> tag = TPM_TAG_STORED_DATA12
      i. Set V1 to 2
      ii. Map S2 a TPM_STORED_DATA12 structure to inData
   b. Else If inData -> ver = 1.1
      i. Set V1 to 1
      ii. Map S2 a TPM_STORED_DATA structure to inData
   c. Else
      i. Return TPM_VERSION
5. Create d1 by decrypting S2 -> encData using the key pointed to by parentHandle
6. Validate d1
   a. d1 MUST be a TPM_SEALED_DATA structure

b. d1 -> tpmProof MUST match TPM_PERMANENT_DATA -> tpmProof
c. Set S2 -> encDataSize to 0
d. Set S2 -> encData to NULL
e. Create h1 the SHA-1 of inData
f. d1 -> storedDigest MUST match h1
g. d1 -> payload MUST be TPM_PT_SEALED
h. Any failure MUST return TPM_NOTSEALED_BLOB

7. The TPM MUST validate authorization to use d1 by checking that the HMAC calculation using d1 -> authData as the shared secret matches the dataAuth. Return TPM_AUTHFAIL on mismatch.
8. If S2 -> pcrInfoSize is not 0 then
   a. If V1 is 1 then
      i. Validate that S2 -> pcrInfo is a valid TPM_PCR_INFO structure
      ii. Create h2 the composite hash of the PCR selected by S2 -> pcrInfo -> pcrSelection
   b. If V1 is 2 then
      i. Validate that S2 -> pcrInfo is a valid TPM_PCR_INFO_LONG structure
      ii. Create h2 the composite hash of the PCR selected by S2 -> pcrInfo -> releasePCRSelection
      iii. Check that S2 -> pcrInfo -> localityAtRelease for TPM_STANY_DATA -> localityModifier is TRUE
         (1) For example if TPM_STANY_DATA -> localityModifier was 2 then S2 -> pcrInfo -> localityAtRelease -> TPM_LOC_TWO would have to be TRUE
   c. Compate h2 with S2 -> pcrInfo -> digestAtRelease, on mismatch return TPM_WRONGPCRVALUE
9. Set the return secret as d1 -> data
10. Return TPM_SUCCESS

# 10.3 TPM_UnBind

***Start of informative comment:***

TPM_UnBind takes the data blob that is the result of a TSS_Bind command and decrypts it for export to the User. The caller must authorize the use of the key that will decrypt the incoming blob.

UnBInd operates on a block-by-block basis, and has no notion of any relation between one block and another.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_UnBind. |
| 4 | 4 | | | TPM_KEY_HANDLE | keyHandle | The keyHandle identifier of a loaded key that can perform UnBind operations. |
| 5 | 4 | 2S | 4 | UINT32 | inDataSize | The size of the input blob |
| 6 | <> | 3S | <> | BYTE[ ] | inData | Encrypted blob to be decrypted |
| 7 | 4 | | | TPM_AUTHHANDLE | authHandle | The handle used for keyHandle authorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | 20 | | | TPM_AUTHDATA | privAuth | The authorization digest that authorizes the inputs and use of keyHandle. HMAC key: key.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_UnBind |
| 4 | 4 | 3S | 4 | UINT32 | outDataSize | The length of the returned decrypted data |
| 5 | <> | 4S | <> | BYTE[ ] | outData | The resulting decrypted data. |
| 6 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth. |

**Description**

UnBind SHALL operate on a single block only.

**Actions**

The TPM SHALL perform the following:

1. If the inDataSize is 0 the TPM returns TPM_BAD_PARAMETER
2. Validate the authorization to use the key pointed to by keyHandle
3. If the keyUsage field of the key referenced by keyHandle does not have the value TPM_KEY_BIND or TPM_KEY_LEGACY, the TPM must return the error code TPM_INVALID_KEYUSAGE
4. Decrypt the inData using the key pointed to by keyHandle
5. if (keyHandle -> encScheme does not equal TPM_ES_RSAESOAEP_SHA1_MGF1) and (keyHandle -> keyUsage equals TPM_KEY_LEGACY),
   a. The payload does not have TPM specific markers to validate, so no consistency check can be performed.
   b. Set the output parameter outData to the value of the decrypted value of inData. (Padding associated with the encryption wrapping of inData SHALL NOT be returned.)
   c. Set the output parameter outDataSize to the size of outData, as deduced from the decryption process.
6. else
   a. Interpret the decrypted data under the assumption that it is a TPM_BOUND_DATA structure, and validate that the payload type is TPM_PT_BIND
   b. Set the output parameter outData to the value of TPM_BOUND_DATA -> payloadData. (Other parameters of TPM_BOUND_DATA SHALL NOT be returned. Padding associated with the encryption wrapping of inData SHALL NOT be returned.)
   c. Set the output parameter outDataSize to the size of outData, as deduced from the decryption process and the interpretation of TPM_BOUND_DATA.
7. Return the output parameters.

# 10.4 TPM_CreateWrapKey

***Start of informative comment:***

The TPM_CreateWrapKey command both generates and creates a secure storage bundle for asymmetric keys.

The newly created key can be locked to a specific PCR value by specifying a set of PCR registers.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CreateWrapKey |
| 4 | 4 | | | TPM_KEY_HANDLE | parentHandle | Handle of a loaded key that can perform key wrapping. |
| 5 | 20 | 2S | 20 | TPM_ENCAUTH | dataUsageAuth | Encrypted usage authorization data for the sealed data. |
| 6 | 20 | 3S | 20 | TPM_ENCAUTH | dataMigrationAuth | Encrypted migration authorization data for the sealed data. |
| 7 | <> | 4S | <> | TPM_KEY | keyInfo | Information about key to be created, pubkey.keyLength and keyInfo.encData elements are 0. MAY be TPM_KEY12 |
| 8 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for parent key authorization. Must be an OS_AP session. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 9 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 10 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Ignored |
| 11 | 20 | | | TPM_AUTHDATA | pubAuth | The authorization digest that authorizes the use of the public key in parentHandle. HMAC key: parentKey.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CreateWrapKey |
| 4 | <> | 4S | <> | TPM_KEY | wrappedKey | The TPM_KEY structure which includes the public and encrypted private key. MAY be TPM_KEY12 |
| 5 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, fixed at FALSE |
| 7 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: parentKey.usageAuth. |

**Descriptions**

This command requires the encryption of two parameters. To create two XOR strings the caller combines the two nonces in use by the OSAP session with the session shared secret.

DataUsageAuth is XOR'd with the SHA-1 hash of the concatenation of the OSAP session shared secret with the even numbered nonce generated by the TPM (authLastNonceEven). MigrationAuth is XOR'd with the SHA-1 hash of the concatenation of the OSAP session shared secret with the odd numbered nonce generated by the caller (nonceOdd).

**Actions**

The TPM SHALL do the following:

1. Validate the authorization to use the key pointed to by parentHandle. Return TPM_AUTHFAIL on any error.
2. Validate the session type for parentHandle is OSAP.
3. If the TPM is not designed to create a key of the type requested in keyInfo, return the error code TPM_BAD_KEY_PROPERTY
4. Verify that parentHandle->keyUsage equals TPM_KEY_STORAGE
5. If parentHandle -> keyFlag -> migratable is TRUE and keyInfo -> keyFlag -> migratable is FALSE then return TPM_INVALID_KEYUSAGE
6. Validate key parameters
   a. keyInfo -> keyUsage MUST NOT be TPM_KEY_IDENTITY or TPM_KEY_AUTHCHANGE. If it is, return TPM_INVALID_KEYUSAGE
   b. If keyInfo -> keyFlags -> migrateAuthority is TRUE then return TPM_INVALID_KEYUSAGE
7. If keyInfo -> keyUsage equals TPM_KEY_STORAGE
   i. algorithmID MUST be TPM_ALG_RSA
   ii. encScheme MUST be TPM_ES_RSAESOAEP_SHA1_MGF1
   iii. sigScheme MUST be TPM_SS_NONE
   iv. key size MUST be 2048

8. Determine the version of key
   a. If keyInfo -> ver is 1.1
      i. Set V1 to 1
      ii. Map wrappedKey to a TPM_KEY structure
   b. Else if keyInfo -> tag is TPM_TAG_KEY12
      i. Set V1 to 2
      ii. Map wrappedKey to a TPM_KEY12 structure
9. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret || authLastNonceEven)
10. Create X2 the SHA-1 of the concatenation of (authHandle -> sharedSecret || nonceOdd)
11. Create DU1 the XOR of DataUsageAuth and X1
12. Create DM1 the XOR of DataMigrationAuth and X2
13. Set continueAuthSession to FALSE
14. Generate asymmetric key according to algorithm information in keyInfo
15. Fill in the wrappedKey structure with information from the newly generated key.
   a. Set wrappedKey -> encData -> usageAuth to DU1
   b. If the KeyFlags -> migratable bit is set to 1, the wrappedKey -> encData -> migrationAuth SHALL contain the decrypted value from DataMigrationAuth.
   c. If the KeyFlags -> migratable bit is set to 0, and wrappedKey -> encData -> migrationAuth SHALL be set to the value tpmProof
16. If wrappedKey->PCRInfoSize is non-zero
   a. If V1 is 1
      i. Set wrappedKey -> pcrInfo to a TPM_PCR_INFO structure using the pcrSelection to indicate the PCR's in use
   b. Else
      i. Set wrappedKey -> pcrInfo to a TPM_PCR_INFO_LONG structure
   c. If keyFlags -> migratable is TRUE
      i. Set digestAtCreation to NULLS
      ii. Set localityAtCreation to NULLS
   d. Else
      i. Set digestAtCreation to the TPM_COMPOSITE_HASH indicated by creationPCRSelection
      ii. Set localityAtCreation to TPM_STANY_DATA -> locality
17. Encrypt the private portions of the wrappedKey structure using the key in keyHandle
18. Return the newly generated key in the wrappedKey parameter

# 10.5 TPM_LoadKey

***Start of informative comment:***

Before the TPM can use a key to either wrap, unwrap, bind, unbind, seal, unseal, sign or perform any other action, it needs to be present in the TPM. The TPM_LoadKey function loads the key into the TPM for further use.

The TPM assigns the key handle. The TPM always locates a loaded key by use of the handle. The assumption is that the handle may change due to key management operations. It is the responsibility of upper level software to maintain the mapping between handle and any label used by external software.

The load command must maintain a record of whether any previous key in the key hierarchy was bound to a PCR using parentPCRStatus.

This command has the responsibility of enforcing restrictions on the use of keys. For example, when attempting to load a STORAGE key it will be checked for the restrictions on a storage key (2048 size etc.).

The flag parentPCRStatus enables the possibility of checking that a platform passed through some particular state or states before finishing in the current state. A grandparent key could be linked to state-1, a parent key could linked to state-2, and a child key could be linked to state-3, for example. The use of the child key then indicates that the platform passed through states 1 and 2 and is currently in state 3, in this example. The issue of TPM_Startup is with stType == TPM_ST_CLEAR is an indication that the platform has been reset, so the platform has not passed through the previous states. Hence keys with parentPCRStatus==TRUE must be unloaded if TPM_Startup is issued with stType == TPM_ST_CLEAR.

If a TPM_KEY structure has been decrypted AND the integrity test using "pubDataDigest" has passed AND the key is non-migratory, the key must have been created by the TPM. So there is every reason to believe that the key poses no security threat to the TPM. While there is no known attack from a rogue migratory key, there is a desire to verify that a loaded migratory key is a real key, arising from a general sense of unease about execution of arbitrary data as a key. Ideally a consistency check would consist of an encrypt/decrypt cycle, but this may be expensive. For RSA keys, it is therefore suggested that the consistency test consists of dividing the supposed RSA product by the supposed RSA prime, and checking that there is no remainder.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_LoadKey. |
| 4 | 4 | | | TPM_KEY_HANDLE | parentHandle | TPM handle of parent key. |
| 5 | <> | 2S | <> | TPM_KEY | inKey | Incoming key structure, both encrypted private and clear public portions. MAY be TPM_KEY12 |
| 6 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for parentHandle authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 7 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 9 | 20 | | | TPM_AUTHDATA | parentAuth | The authorization digest for inputs and parentHandle. HMAC key: parentKey.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_LoadKey |
| 4 | 4 | 3S | 4 | TPM_KEY_HANDLE | inkeyHandle | Internal TPM handle where decrypted key was loaded. |
| 5 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 7 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: parentKey.usageAuth. |

**Actions**

The TPM SHALL perform the following steps:

1. Validate the command and the parameters using parentAuth and parentHandle -> useAuth
2. If parentHandle -> keyUsage is NOT TPM_KEY_STORAGE return TPM_INVALID_KEYUSAGE
3. If the TPM is not designed to operate on a key of the type specified by inKey, return the error code TPM_BAD_KEY_PROPERTY
4. The TPM MUST handle both TPM_KEY and TPM_KEY12 structures
5. Decrypt the inKey -> privkey to obtain TPM_STORE_ASYMKEY structure using the key in parentHandle
6. Validate the integrity of inKey and decrypted TPM_STORE_ASYMKEY
   a. Reproduce inKey -> TPM_STORE_ASYMKEY -> pubDataDigest using the fields of inKey, and check that the reproduced value is the same as pubDataDigest
7. Validate the consistency of the key and it's key usage.
   a. If inKey -> keyFlags -> migratable is TRUE, the TPM SHALL verify consistency of the public and private components of the asymmetric key pair. If inKey -> keyFlags -> migratable is FALSE, the TPM MAY verify consistency of the public and private components of the asymmetric key pair. The consistency of an RSA key pair MAY be verified by dividing the supposed (P*Q) product by a supposed prime and checking that there is no remainder..
   b. If inKey -> keyUsage is TPM_KEY_IDENTITY, verify that inKey->keyFlags->migratable is FALSE. If it is not, return TPM_INVALID_KEYUSAGE
   c. If inKey -> keyUsage is TPM_KEY_AUTHCHANGE, return TPM_INVALID_KEYUSAGE
   d. If inKey -> keyFlags -> migratable equals 0 then verify that TPM_STORE_ASYMKEY -> migration equals TPM_PERSISTENT_DATA -> tpmProof
   e. Validate the mix of encryption and signature schemes
   f. If inKey -> keyUsage is TPM_KEY_STORAGE
      i. algorithmID MUST be TPM_ALG_RSA
      ii. Key size MUST be 2048
      iii. sigScheme MUST be TPM_SS_NONE

g. If inKey -> keyUsage is TPM_KEY_IDENTITY
   i. algorithmID MUST be TPM_ALG_RSA
   ii. Key size MUST be 2048
   iii. encScheme MUST be TPM_ES_NONE
h. If the decrypted inKey -> pcrInfo is NULL,
   i. The TPM MUST set the internal indicator to indicate that the key is not using any PCR registers.
i. Else
   i. The TPM MUST store pcrInfo in a manner that allows the TPM to calculate a composite hash whenever the key will be in use
   ii. The TPM MUST handle both version 1.1 TPM_PCR_INFO and 1.2 TPM_PCR_INFO_LONG structures according to the type of TPM_KEY structure

8. Perform any processing necessary to make TPM_STORE_ASYMKEY key available for operations
9. Load key and key information into internal memory of the TPM. If insufficient memory exists return error TPM_NOSPACE.
10. Assign inKeyHandle according to internal TPM rules.
11. Set InKeyHandle -> parentPCRStatus to parentHandle -> parentPCRStatus.
12. If ParentHandle indicates it is using PCR registers then set inKeyHandle -> parentPCRStatus to TRUE.

# 10.6 TPM_GetPubKey

***Start of informative comment:***

The owner of a key may wish to obtain the public key value from a loaded key. This information may have privacy concerns so the command must have authorization from the key owner.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_GetPubKey. |
| 4 | 4 | | | TPM_KEY_HANDLE | keyHandle | TPM handle of key. |
| 5 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 6 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 8 | 20 | | | TPM_AUTHDATA | keyAuth | Authorizatoin HMAC key: key.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_GetPubKey. |
| 4 | <> | 3S | <> | TPM_PUBKEY | pubKey | Public portion of key in keyHandle. |
| 5 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 7 | 20 | | | TPM_AUTHDATA | resAuth | Authorization. HMAC key: key.usageAuth. |

**Actions**

The TPM SHALL perform the following steps:

1. Validate the command the parameters using keyAuth, on error
   a. If keyHandle has TPM_AUTH_PRIV_USE_ONLY ignore the error
   b. Otherwise return TPM_AUTHFAIL
2. If keyHandle == TPM_KH_SRK then return TPM_INVALID_PARAMETER
3. If keyHandle -> pcrInfoSize is not 0
   a. If keyHandle -> keyFlags has pcrIgnoredOnRead set to FALSE

i. Create a digestAtRelease according to the specified PCR registers and compare to keyHandle -> digestAtRelease and if a mismatch return TPM_WRONGPCRVAL

ii. If specified validate any locality requests

4. Create a TCG_PUBKEY structure and return

# 11. Migration

## 11.1 TPM_CreateMigrationBlob

***Start of informative comment:***

The TPM_CreateMigrationBlob command implements the first step in the process of moving a migratable key to a new parent or platform. Execution of this command requires knowledge of the migrationAuth field of the key to be migrated.

Migrate mode is generally used to migrate keys from one TPM to another for backup, upgrade or to clone a key on another platform. To do this, the TPM needs to create a data blob that another TPM can deal with. This is done by loading in a backup public key that will be used by the TPM to create a new data blob for a migratable key.

The TPM Owner does the selection and authorization of migration public keys at any time prior to the execution of TPM_CreateMigrationBlob by performing the TPM_AuthorizeMigrationKey command.

IReWrap mode is used to directly move the key to a new parent (either on this platform or another). The TPM simply re-encrypts the key using a new parent, and outputs a normal encrypted element that can be subsequently used by a TPM_LoadKey command.

TPM_CreateMigrationBlob implicitly cannot be used to migrate a non-migratory key. No explicit check is required. Only the TPM knows tpmProof. Therefore it is impossible for the caller to submit an authorization value equal to tpmProof and migrate a non-migratory key.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CreateMigrationBlob |
| 4 | 4 | | | TPM_KEY_HANDLE | parentHandle | Handle of the parent key that can decrypt encData. |
| 5 | 2 | 2S | 2 | TPM_MIGRATE_SCHEME | migrationType | The migration type, either MIGRATE or REWRAP |
| 6 | <> | 3S | <> | TPM_MIGRATIONKEYAUTH | migrationKeyAuth | Migration public key and its authorization digest. |
| 7 | 4 | 4S | 4 | UINT32 | encDataSize | The size of the encData parameter |
| 8 | <> | 5S | <> | BYTE[ ] | encData | The encrypted entity that is to be modified. |
| 9 | 4 | | | TPM_AUTHHANDLE | parentAuthHandle | The authorization handle used for the parent key. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 10 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with parentAuthHandle |
| 11 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag for parent session |
| 12 | 20 | | 20 | TPM_AUTHDATA | parentAuth | Authorization HMAC key: parentKey.usageAuth. |
| 13 | 4 | | | TPM_AUTHHANDLE | entityAuthHandle | The authorization handle used for the encrypted entity. |
| | | 2H2 | 20 | TPM_NONCE | entitylastNonceEven | Even nonce previously generated by TPM |
| 14 | 20 | 3H2 | 20 | TPM_NONCE | entitynonceOdd | Nonce generated by system associated with entityAuthHandle |
| 15 | 1 | 4H2 | 1 | BOOL | continueEntitySession | Continue use flag for entity session |

| | | | | | | |
|---|---|---|---|---|---|---|
| 16 | 20 | | | TPM_AUTHDATA | entityAuth | Authorization HMAC key: entity.migrationAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CreateMigrationBlob |
| 4 | 4 | 3S | 4 | UINT32 | randomSize | The used size of the output area for random |
| 5 | <> | 4S | <> | BYTE[ ] | random | String used for xor encryption |
| 6 | 4 | 5S | 4 | UINT32 | outDataSize | The used size of the output area for outData |
| 7 | <> | 6S | <> | BYTE[ ] | outData | The modified, encrypted entity. |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 4H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with parentAuthHandle |
| 9 | 1 | 5H1 | 1 | BOOL | continueAuthSession | Continue use flag for parent key session |
| 10 | 20 | | 20 | TPM_AUTHDATA | resAuth | Authorization. HMAC key: parentKey.usageAuth. |
| 11 | 20 | 3H2 | 20 | TPM_NONCE | entityNonceEven | Even nonce newly generated by TPM to cover entity |
| | | 4H2 | 20 | TPM_NONCE | entitynonceOdd | Nonce generated by system associated with entityAuthHandle |
| 12 | 1 | 5 H2 | 1 | BOOL | entityContinueAuthSession | Continue use flag for entity session |
| 13 | 20 | | | TPM_AUTHDATA | entityAuth | Authorization HMAC key: entity.migrationAuth. |

**Description**

The TPM does not check the PCR values when migrating values locked to a PCR.

The second authorisation session (using entityAuth) MUST be OIAP because OSAP does not have a suitable entityType

**Actions**

1. Validate that parentAuth authorizes the use of the key pointed to by parentHandle.
2. Create d1 by decrypting encData using the key pointed to by parentHandle.
3. Validate that entityAuth authorizes the migration of d1. The validation MUST use d1 -> migrationAuth as the secret.
4. Verify that the digest within migrationKeyAuth is legal for this TPM and public key
5. If migrationType == TPM_MS_MIGRATE the TPM SHALL perform the following actions:
   a. Build a TPM_STORE_PRIVKEY structure from the d1 key. This privKey element should be 132 bytes long for a 2K RSA key.
   b. Create k1 and k2 by splitting the privKey element created in step a into 2 parts. k1 is the first 20 bytes of privKey, k2 contains the remainder of privKey.
   c. Build m by filling in the usageAuth and pubDataDigest fields within a TPM_MIGRATE_ASYMKEY structure using data from the d1 key. The privKey field should be set to k2 (step g) and payload should be set to TPM_PT_MIGRATE.

d. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP encoding of m using OAEP parameters of
   i. *m* = TPM_MIGRATE_ASYMKEY structure (step c)
   ii. *pHash* = d1->migrationAuth
   iii. *seed* = *s1* = k1 (step g)
e. Create r1 a random value from the TPM RNG. The size of r1 MUST be the size of o1. Return r1 in the Random parameter.
f. Create x1 by XOR of o1 with r1
g. Copy r1 into the output field “random”.
h. Encrypt x1 with the migration public key included in migrationKeyAuth.

6. If migrationType == TPM_MS_REWRAP the TPM SHALL perform the following actions:
   a. Rewrap the key using the public key in migrationKeyAuth, keeping the existing contents of that key.
   b. Set randomSize to 0 in the output parameter array

# 11.2 TPM_ConvertMigrationBlob

***Start of informative comment:***

This command takes a migration blob and creates a normal wrapped blob. The migrated blob must be loaded into the TPM using the normal TPM_LoadKey function.

Note that the command migrates private keys, only. The migration of the associated public keys is not specified by TPM because they are not security sensitive. Migration of the associated public keys may be specified in a platform specific specification. A TPM_KEY structure must be recreated before the migrated key can be used by the target TPM in a LoadKey command.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ConvertMigrationBlob. |
| 4 | 4 | | | TPM_KEY_HANDLE | parentHandle | Handle of a loaded key that can decrypt keys. |
| 5 | 4 | 2S | 4 | UINT32 | inDataSize | Size of inData |
| 6 | <> | 3S | <> | BYTE [ ] | inData | The XOR'd and encrypted key |
| 7 | 4 | 4S | 4 | UINT32 | randomSize | Size of random |
| 8 | <> | 5S | <> | BYTE [ ] | random | Random value used to hide key data. |
| 9 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for keyHandle. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 10 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 11 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 12 | 20 | | | TPM_AUTHDATA | parentAuth | The authorization digest that authorizes the inputs and the migration of the key in parentHandle. HMAC key: parentKey.usageAuth |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ConvertMigrationBlob |
| 4 | 4 | 3S | 4 | UINT32 | outDataSize | The used size of the output area for outData |
| 5 | <> | 4S | <> | BYTE[ ] | outData | The encrypted private key that can be loaded with TPM_LoadKey |
| 6 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: parentKey.usageAuth |

**Action**

The TPM SHALL perform the following:

1. Validate the authorization to use the key in parentHandle
2. If the keyUsage field of the key referenced by parentHandle does not have the value TPM_KEY_STORAGE, the TPM must return the error code TPM_INVALID_KEYUSAGE
3. Create d1 by decrypting the inData area using the key in parentHandle
4. Create o1 by XOR d1 and random parameter
5. Create m1, seed and pHash by OAEP decoding o1
6. Verify that the payload type is TPM_PT_MIGRATE
7. Create k1 by combining seed and the TPM_MIGRATE_ASYMKEY.data field
8. Create d2 a TPM_STORE_ASYMKEY structure by inserting pHash as the migration authorization field. Set the TPM_STORE_ASYMKEY -> privKey field to k1
9. Create outData using the key in parentHandle to perform the encryption

# 11.3 TPM_AuthorizeMigrationKey

***Start of informative comment:***

This command creates an authorization blob, to allow the TPM owner to specify which migration facility they will use and allow users to migrate information without further involvement with the TPM owner.

It is the responsibility of the TPM Owner to determine whether migrationKey is appropriate for migration. The TPM checks just the cryptographic strength of migrationKey.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed at TPM_ORD_AuthorizeMigrationKey |
| 4 | 2 | 2S | 2 | TPM_MIGRATE_SCHEME | migrateScheme | Type of migration operation that is to be permitted for this key. |
| 4 | <> | 3S | <> | TPM_PUBKEY | migrationKey | The public key to be authorized. |
| 5 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 6 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 8 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed at TPM_ORD_AuthorizeMigrationKey |
| 4 | <> | 3S | <> | TPM_MIGRATIONKEYAUTH | outData | Returned public key and authorization digest. |
| 5 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 7 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Action**

The TPM SHALL perform the following:

1. Check that the cryptographic strength of migrationKey is at least that of a 2048 bit RSA key. If migrationKey is an RSA key, this means that migrationKey MUST be 2048 bits or greater

2. Validate the authorization to use the TPM by the TPM Owner
3. Create a f1 a TPM_MIGRATIONKEYAUTH structure
4. Verify that migrationKey-> algorithmParms -> encScheme is TPM_ES_RSAESOAEP_SHA1_MGF1, and return the error code TPM_INAPPROPRIATE_ENC if it is not
5. Set f1 -> migrationKey to the input migrationKey
6. Set f1 -> migrationScheme to the input migrationScheme
7. Create v1 by concatenating (migrationKey || migrationScheme || TPM_PERMANENT_DATA -> tpmProof)
8. Create h1 by performing a SHA1 hash of v1
9. Set f1 -> digest to h1
10. Return f1 as outData

# 11.4 TPM_CMK_CreateKey

***Start of informative comment:***

The TPM_CreateWrapRestrictedKey command both generates and creates a secure storage bundle for asymmetric keys whose migration is controlled by a migration authority.

TPM_CreateWrapRestrictedKey is very similar to TPM_CreateWrapKey, but: (1) the resultant key must be a migratable key and can be migrated only by TPM_CMK_CreateBlob; (2) the command is Owner authorised. TPM_CreateWrapRestrictedKey creates an otherwise normal key except that migrationAuth is the digest of tpmProof, the migration authority, and the new key's public key (instead of a secret value).

The migration-selection/migration authority is specified by passing in a public key (actually the digest of a public key).

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CMK_CreateKey |
| 4 | 4 | | | TPM_KEY_HANDLE | parentHandle | Handle of a loaded key that can perform key wrapping. |
| 5 | 20 | 2S | 20 | TPM_ENCAUTH | dataUsageAuth | Encrypted usage authorization data for the sealed data. |
| 6 | <> | 3S | <> | TPM_KEY | keyInfo | Information about key to be created, pubkey.keyLength and keyInfo.encData elements are 0. MUST be TPM_KEY12 |
| 7 | 20 | 4S | 20 | TPM_DIGEST | migrationAuthorityDigest | The digest of the public key of the MSA or MA |
| 8 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for parent key authorization. Must be an OS_AP session. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 9 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 10 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Ignored |
| 11 | 20 | | | TPM_AUTHDATA | pubAuth | The authorization digest that authorizes the use of the public key in parentHandle. HMAC key: parentKey.usageAuth. |
| 12 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H2 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 13 | 20 | 3H2 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 14 | 1 | 4H2 | 1 | BOOL | continueAuthSession | Ignored |
| 15 | 20 | | | TPM_AUTHDATA | pubAuth | The authorization digest for nputs and owner. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CMK_CreateKey |
| 4 | <> | 3S | <> | TPM_KEY | wrappedKey | The TPM_KEY structure which includes the public and encrypted private key. MUST be TPM_KEY12 |
| 5 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, fixed at FALSE |
| 7 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: parentKey.usageAuth. |
| 8 | 20 | 2H2 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H2 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H2 | 1 | BOOL | continueAuthSession | Continue use flag, fixed at FALSE |
| 10 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key:. ownerAuth. |

**Actions**

The TPM SHALL do the following:

1. Validate the authorization to use the key pointed to by parentHandle. Return TPM_AUTHFAIL on any error.
2. Validate the session type for parentHandle is OSAP.
3. If the TPM is not designed to create a key of the type requested in keyInfo, return the error code TPM_BAD_KEY_PROPERTY
4. Verify that parentHandle->keyUsage equals TPM_KEY_STORAGE
5. Verify that parentHandle-> keyFlags-> migratable == FALSE and parentHandle-> encData -> migrationAuth == tpmProof
6. If keyInfo -> keyFlag -> migratable is FALSE then return TPM_INVALID_KEYUSAGE
7. Validate key parameters
   a. keyInfo -> keyUsage MUST NOT be TPM_KEY_IDENTITY or TPM_KEY_AUTHCHANGE. If it is, return TPM_INVALID_KEYUSAGE
8. If keyInfo -> keyUsage equals TPM_KEY_STORAGE
   i. algorithmID MUST be TPM_ALG_RSA
   ii. encScheme MUST be TPM_ES_RSAESOAEP_SHA1_MGF1
   iii. sigScheme MUST be TPM_SS_NONE
   iv. key size MUST be 2048
9. If keyinfo -> tag is NOT TPM_TAG_KEY12 return TPM_INVALID_STRUCTURE

10. Map wrappedKey to a TPM_KEY12 structure
11. Create X1 the SHA-1 of the concatenation of (authHandle -> sharedSecret || authLastNonceEven)
12. Create DU1 the XOR of DataUsageAuth and X1
13. Set continueAuthSession to FALSE
14. Generate asymmetric key according to algorithm information in keyInfo
15. Fill in the wrappedKey structure with information from the newly generated key.
    a. Set wrappedKey -> encData -> usageAuth to DU1
    b. Set wrappedKey -> encData -> payload to TPM_PT_MIGRATE_RESTRICTED
    c. Create thisPubKey, a TPM_PUBKEY structure containing wrappedKey's public key.
    d. Set wrappedKey -> encData -> migrationAuth equal to SHA1(tpmProof || migrationAuthorityDigest || thisPubKey)
16. If wrappedKey->PCRInfoSize is non-zero
    a. Set wrappedKey -> pcrInfo to a TPM_PCR_INFO_LONG structure
    b. Set digestAtCreation to the TPM_COMPOSITE_HASH indicated by creationPCRSelection
    c. Set localityAtCreation to TPM_STANY_DATA -> locality
17. Encrypt the private portions of the wrappedKey structure using the key in parentHandle
18. Return the newly generated key in the wrappedKey parameter

## 11.5 TPM_CMK_CreateTicket

***Start of informative comment:***

The TPM_verifySignature command uses a public key to verify the signature over a digest.

TPM_verifySignature returns a ticket that can be used to prove to the same TPM that signature verification with a particular public key was successful.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CMK_CreateTicket |
| 4 | 4 | | | TPM_PUBKEY | verificationKey | The public key to be used to check signatureValue |
| 5 | 20 | 2S | 20 | TPM_DIGEST | signedData | The data proported to be signed |
| 6 | 4 | 3S | 4 | UINT32 | signatureValueSize | The size of the signatureValue |
| 7 | <> | 4S | <> | BYTE[] | signatureValue | The signatureValue to be verified |
| 8 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 9 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 10 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Ignored |
| 11 | 20 | | | TPM_AUTHDATA | pubAuth | The authorization digest for nputs and owner. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CMK_CreateTicket |
| 4 | 20 | 3S | 20 | TPM_DIGEST | sigTicket | Ticket that proves digest created on this TPM |
| 5 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag |
| 7 | 20 | | | TPM_AUTHDATA | resAuth | Authorization. HMAC key:. ownerAuth. |

**Actions**

The TPM SHALL do the following:

1. Validate the TPM Owner authorization to use the command
2. Use verificationKey to verify that signatureValue is a valid signature on signedData, and return error TPM_BAD_SIGNATURE on mismatch
3. set sigTicket = SHA1(tpmProof || SHA1(verificationKey) || signedData)
4. Return TPM_SUCCESS

## 11.6 TPM_CMK_CreateBlob

***Start of informative comment:***

TPM_CMK_CreateBlob command is very similar to TPM_CreateMigrationBlob, except that it: (1) uses an extra ticket (restrictedKeyAuth) instead of a migrationAuth authorization session; (2) uses the migration options TPM_MS_RESTRICT_MIGRATE or TPM_MS_RESTRICT_APPROVE.

If the public key in migrationKeyAuth matches migrationAuth in the target key-to-be-migrated, the target can be migrated without use of restrictedMigrationAuth. Otherwise, the ticket "restrictedKeyAuth" must vouch for the public key in migrationKeyAuth.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CMK_CreateBlob |
| 4 | 4 | | | TPM_KEY_HANDLE | parentHandle | Handle of the parent key that can decrypt encData. |
| 5 | 2 | 2S | 2 | TPM_MIGRATE_SCHEME | migrationType | The migration type, either TPM_MS_RESTRICT_MIGRATE or TPM_MS_RESTRICT_APPROVE |
| 6 | <> | 3S | <> | TPM_MIGRATIONKEYAUTH | migrationKeyAuth | Migration public key and its authorization digest. |
| 7 | 20 | 4S | 20 | TPM_DIGEST | pubSourceKeyDigest | The digest of the TPM_PUBKEY of the entity to be migrated |
| 8 | 4 | 5S | 4 | UINT32 | restrictTicketSize | The size of the restrictTicket parameter |
| 9 | <> | 6S | <> | BYTE[] | restrictTicket | The digests of public keys belonging to the Migration Authorit, the destination parent key and the key-to-be-migrated. |
| 10 | 4 | 7S | 4 | UINT32 | sigTicketSize | The size of the sigTicket parameter |
| 11 | <> | 8S | <> | BYTE[] | sigTicket | A signature ticket, generate by the TPM, signalling a valid signature over restrictTicket |
| 12 | | 9S | | UINT32 | encDataSize | The size of the encData parameter |
| 13 | | 10S | | BYTE[] | encData | The encrypted entity that is to be modified. |
| 14 | 4 | | | TPM_AUTHHANDLE | parentAuthHandle | The authorization handle used for the parent key. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 15 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with parentAuthHandle |
| 16 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag for parent session |
| 17 | 20 | | 20 | TPM_AUTHDATA | parentAuth | The authorization digest for inputs and parentHandle. HMAC key: parentKey.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CMK_CreateBlob |
| 4 | 4 | 3S | 4 | UINT32 | randomSize | The used size of the output area for random |
| 5 | <> | 4S | <> | BYTE[ ] | random | String used for xor encryption |
| 6 | 4 | 5S | 4 | UINT32 | outDataSize | The used size of the output area for outData |
| 7 | <> | 6S | <> | BYTE[ ] | outData | The modified, encrypted entity. |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 4H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with parentAuthHandle |
| 9 | 1 | 5H1 | 1 | BOOL | continueAuthSession | Continue use flag for parent key session |
| 10 | 20 | | 20 | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters and parentHandle. HMAC key: parentKey.usageAuth. |

**Description**

The TPM does not check the PCR values when migrating values locked to a PCR.

**Actions**

1. Validate that parentAuth authorizes the use of the key pointed to by parentHandle.
2. Verify that parentHandle-> keyFlags-> migratable == FALSE and parentHandle-> encData -> migrationAuth == tpmProof
3. Create d1 by decrypting encData using the key pointed to by parentHandle.
4. Verify that the digest within migrationKeyAuth is legal for this TPM and public key
5. Verify that d1 -> payload == TPM_PT_MIGRATE_RESTRICTED
6. Verify that migrationKeyAuth -> migrationScheme == TPM_MS_RESTRICT_MIGRATE or TPM_MS_RESTRICT_APPROVE
7. If migrationKeyAuth -> migrationScheme == TPM_MS_RESTRICT_MIGRATE
   a. Verify that [d1 -> migrationAuth] == SHA1(tpmProof || (SHA1[migrationKeyAuth -> migrationKey]) || pubSourceKeyDigest) and return error TPM_MA_AUTHORITY on mismatch
8. If migrationKeyAuth -> migrationScheme == TPM_MS_RESTRICT_APPROVE or TPM_MS_RESTRICT_APPROVE_DOUBLE
   a. verify that [d1 -> migrationAuth] == SHA1(tpmProof || (restrictTicket -> migrationAuthorityDigest) || (restrictTicket -> sourceKeyDigest) ) and return error TPM_MA_AUTHORITY on mismatch
   b. verify that [restrictTicket -> destinationKeyDigest] == SHA1[migrationKeyAuth -> migrationKey] and return error TPM_MA_DESTINATION on mismatch
   c. verify that [restrictTicket -> sourceKeyDigest] == pubSourceKeyDigest and return error TPM_MA_SOURCE on mismatch

d. verify that sigTicket == SHA1( tpmProof || (restrictTicket -> migrationAuthorityDigest) || SHA1(restrictTicket) ) and return error TPM_MA_TICKET_SIGNATURE on mismatch

9. If migrationType == TPM_MS_RESTRICT_MIGRATE or TPM_MS_RESTRICT_APPROVE_DOUBLE, the TPM SHALL perform the following actions:
   a. Build a TPM_STORE_PRIVKEY structure from the d1 key. This privKey element should be 132 bytes long for a 2K RSA key.
   b. Create k1 and k2 by splitting the privKey element created in step a into 2 parts. k1 is the first 20 bytes of privKey, k2 contains the remainder of privKey.
   c. Build m by filling in the usageAuth and pubDataDigest fields within a TPM_MIGRATE_ASYMKEY structure using data from the d1 key. The privKey field should be set to k2 (step g) and payload should be set to TPM_PT_MIGRATE.
   d. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP encoding of m using OAEP parameters of
      i. *m* = TPM_MIGRATE_ASYMKEY structure (step c)
      ii. *pHash* = d1->migrationAuth
      iii. *seed* = *s1* = k1 (step g)
   e. Create r1 a random value from the TPM RNG. The size of r1 MUST be the size of o1. Return r1 in the Random parameter.
   f. Create x1 by XOR of o1 with r1
   g. Copy r1 into the output field “random”.
   h. Encrypt x1 with the migrationKeyAuth-> migrationKey
10. If migrationType == TPM_MS_RESTRICT_APPROVE the TPM SHALL perform the following actions:
   a. Rewrap the key using the public key in migrationKeyAuth, keeping the existing contents of that key.
   b. Set randomSize to 0 in the output parameter array

# 11.7 TPM_CMK_SetRestrictions

***Start of informative comment:***

This command is used by the Owner to dictate the usage of a restricted-migration key with delegated authorisation (authorisation other than actual Owner authorisation).

This command is provided for privacy reasons, since creation of a "parent" restricted-migration key may imply a contractual relationship with an external entity. This command cannot be delegated, because it controls an aspect of delegation.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes incl. paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Ordinal: TPM_ORD_CMK_SetRestrictions |
| 4 | 4 | 2S | 4 | TPM_CMK_RESTRICTDELEGATE | restriction | The bit mask of how to set the restrictions on CMK keys |
| 5 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle TPM Ownerauthorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 6 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 8 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest. HMAC key:TPM Owner Auth |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Ordinal: TPM_ORD_CMK_SetRestrictions |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | Authorization HMAC key: ownerAuth. |

**Actions**

1. If TPM Owner is installed
   a. Validate the ordinal and parameters using TPM Owner authorization, return TPM_AUTHFAIL on error
2. Else
   a. If no assertion of physical presence the TPM MUST return TPM_BAD_PRESENCE
3. Set TPM_PERMANENT_DATA -> restrictMigrateDelegate = restriction
4. Return TPM_SUCCESS

# 12. Maintenance Functions (optional)

## 12.1 TPM_CreateMaintenanceArchive

***Start of informative comment:***

This command creates the MaintenanceArchive. It can only be executed by the owner, and may be shut off with the TPM_KillMaintenanceFeature command.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Cmd ordinal: TPM_ORD_CreateMaintenanceArchive |
| 4 | 1 | 2S | 1 | BOOL | generateRandom | Use RNG or Owner auth to generate 'random'. |
| 5 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 6 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 8 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Cmd ordinal: TPM_ORD_CreateMaintenanceArchive |
| 4 | 4 | 3S | 4 | UINT32 | randomSize | Size of the returned random data. Will be 0 if generateRandom is FALSE. |
| 5 | <> | 4S | <> | BYTE [ ] | random | Random data to XOR with result. |
| 6 | 4 | 5S | 4 | UINT32 | archiveSize | Size of the encrypted archive |
| 7 | <> | 6S | <> | BYTE [ ] | archive | Encrypted key archive. |
| 8 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 10 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Actions**

Upon authorization being confirmed this command does the following:

1. Validates that the TPM_PERMANENT_FLAGS -> AllowMaintenance is TRUE. If it is FALSE, the TPM SHALL return TPM_DISABLED_CMD and exit this capability.
2. Validates the TPM Owner authorization.
3. If the value of TPM_PERMANENT_DATA -> ManuMaintPub is zero, the TPM MUST return the error code TPM_KEYNOTFOUND
4. Build a1 a TPM_KEY structure using the SRK. The encData field is not a normal TPM_STORE_ASYMKEY structure but rather a TPM_MIGRATE_ASYMKEY structure built using the following actions.
5. Build a TPM_STORE_PRIVKEY structure from the SRK. This privKey element should be 132 bytes long for a 2K RSA key.
6. Create k1 and k2 by splitting the privKey element created in step 4 into 2 parts. k1 is the first 20 bytes of privKey, k2 contains the remainder of privKey.
7. Build m1 by creating and filling in a TPM_MIGRATE_ASYMKEY structure
   a. m1 -> usageAuth is set to TPM_PERSISTENT_FIELDS -> tmpProof
   b. m1 -> pubDataDigest is set to the digest value of the SRK fields from step 4
   c. m1 -> payload is set to TPM_PT_MAINT
   d. m1 -> partPrivKey is set to k2
8. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP encoding of m using OAEP parameters of
   a. *m* = TPM_MIGRATE_ASYMKEY structure (step 7)
   b. *P* = TPM_PERSISTENT_FIELDS -> ownerAuth
   c. *seed* = *s1* = k1 (step 6)
9. If GenerateRandom = TRUE
   a. Create r1 by obtaining values from the TPM RNG. The size of r1 MUST be the same size as o1. Set RandomData parameter to r1
10. If GenerateRandom = FALSE
    a. Create r1 by applying MGF1 to the TPM Owner authorization data. The size of r1 MUST be the same size as o1. Set RandomData parameter to null.
11. Create x1 by XOR of o1 with r1
12. Encrypt x1 with the ManuMaintPub key using the TPM_ES_RSAESOAEP_SHA1_MGF1 encryption scheme.
13. Set a1 -> encData to x1
14. Return a1 in the archive parameter

# 12.2 TPM_LoadMaintenanceArchive

***Start of informative comment:***

This command loads in a Maintenance archive that has been massaged by the manufacturer to load into another TPM

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_LoadMaintenanceArchive |
| | | | | … | … | Vendor specific arguments |
| - | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| - | 20 | | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| - | 1 | | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| -- | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_LoadMaintenanceArchive |
| | | | | .. | .. | Vendor specific arguments |
| - | 20 | | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| - | 1 | | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| - | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Descriptions**

The maintenance mechanisms in the TPM MUST not require the TPM to hold a global secret. The definition of global secret is a secret value shared by more than one TPM.

The TPME is not allowed to pre-store or use unique identifiers in the TPM for the purpose of maintenance. The TPM MUST NOT use the endorsement key for identification or encryption in the maintenance process. The maintenance process MAY use a TPM Identity to deliver maintenance information to specific TPM's.

The maintenance process can only change the SRK, tpmProof and TPM Owner authorization fields.

The maintenance process can only access data in shielded locations where this data is necessary to validate the TPM Owner, validate the TPME and manipulate the blob

The TPM MUST be conformant to the TPM specification, protection profiles and security targets after maintenance. The maintenance MAY NOT decrease the security values from the original security target.

The security target used to evaluate this TPM MUST include this command in the TOE.

**Actions**

The TPM SHALL perform the following when executing the command

1. Validate the TPM Owner's authorization
2. Validate that the maintenance information was sent by the TPME. The validation mechanism MUST use a strength of function that is at least the same strength of function as a digital signature performed using a 2048 bit RSA key.
3. The packet MUST contain m2 as defined in TODOREF
4. Ensure that only the target TPM can interpret the maintenance packet. The protection mechanism MUST use a strength of function that is at least the same strength of function as a digital signature performed using a 2048 bit RSA key.
5. Process the maintenance information and update the SRK and TPM_PERMANENT_DATA -> tpmProof fields.
6. Set the SRK useageAuth to be the same as TPM Owners authorization

# 12.3 TPM_KillMaintenanceFeature

***Informative Comments:***

The KillMaintencanceFeature is a permanent action that prevents ANYONE from creating a maintenance archive. This action, once taken, is permanent until a new TPM Owner is set.

This action is to allow those customers who do not want the maintenance feature to not allow the use of the maintenance feature.

At the discretion of the Owner, it should be possible to kill the maintenance feature in such a way that the only way to recover maintainability of the platform would be to wipe out the root keys. This feature is mandatory in any TPM that implements the maintenance feature.

***End informative Comment***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_KillMaintenanceFeature |
| 4 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 5 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 7 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_KillMaintenanceFeature |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Actions**

1. Validate the TPM Owner authorization
2. Set the TPM_PERMANENT_FLAGS.AllowMaintenance flag to FALSE.

# 12.4 TPM_LoadManuMaintPub

***Informative Comments:***

The LoadManuMaintPub command loads the manufacturer's public key for use in the maintenance process. The command installs ManuMaintPub in persistent data storage inside a TPM. Maintenance enables duplication of non-migratory data in protected storage. There is therefore a security hole if a platform is shipped before the maintenance public key has been installed in a TPM.

The command is expected to be used before installation of a TPM Owner or any key in TPM protected storage. It therefore does not use authorization.

***End of Informative Comments***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_LoadManuMaintPub |
| 4 | 20 | | | TPM_NONCE | antiReplay | AntiReplay and validation nonce |
| 5 | <> | | | TPM_PUBKEY | pubKey | The public key of the manufacturer to be in use for maintenance |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| | | | | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_LoadManuMaintPub |
| 4 | 20 | | | TPM_DIGEST | checksum | Digest of pubKey and antiReplay |

**Description**

The pubKey MUST specify an algorithm whose strength is not less than the RSA algorithm with 2048bit keys.

pubKey SHOULD unambiguously identify the entity that will perform the maintenance process with the TPM Owner.

TPM_PERMANENT_DATA -> ManuMaintPub SHALL exist in a TPM-shielded location, only.

If an entity (Platform Entity) does not support the maintenance process but issues a platform credential for a platform containing a TPM that supports the maintenance process, the value of TPM_PERMANENT_DATA -> ManuMaintPub MUST be set to zero before the platform leaves the entity's control.

**Actions**

The first valid TPM_LoadManuMaintPub command received by a TPM SHALL

1. Store the parameter pubKey as TPM_PERMANENT_DATA -> ManuMaintPub.

2. Create “checksum” by concatenating data to form (pubKey||antiReplay) and passing the concatenated data through a SHA-1 hash process.
3. Export the checksum
4. Subsequent calls to TPM_LoadManuMaintPub SHALL return code TPM_DISABLED_CMD.

# 12.5 TPM_ReadManuMaintPub

***Informative Comments:***

The ReadManuMaintPub command is used to check whether the manufacturer's public maintenance key in a TPM has the expected value. This may be useful during the manufacture process. The command returns a digest of the installed key, rather than the key itself. This hinders discovery of the maintenance key, which may (or may not) be useful for manufacturer privacy.

The command is expected to be used before installation of a TPM Owner or any key in TPM protected storage. It therefore does not use authorization.

***End of Informative Comments***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ReadManuMaintPub |
| 4 | 20 | | | TPM_NONCE | antiReplay | AntiReplay and validation nonce |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| | | | | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ReadManuMaintPub |
| 4 | 20 | | | TPM_DIGEST | checksum | Digest of pubKey and antiReplay |

**Description**

This command returns the hash of the antiReplay nonce and the previously loaded manufacturer's maintenance public key.

**Actions**

The TPM_ ReadManuMaintKey command SHALL

1. Create "checksum" by concatenating data to form (TPM_PERMANENT_DATA -> ManuMaintPub ||antiReplay) and passing the concatenated data through SHA1.
2. Export the checksum

# 13. Cryptographic Functions

## 13.1 TPM_SHA1Start

***Start of informative comment:***

This capability starts the process of calculating a SHA-1 digest.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SHA1Start |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 4 | | | UINT32 | maxNumBytes | Maximum number of bytes that can be sent to TPM_SHA1Update. Must be a multiple of 64 bytes. |

**Description**

This capability prepares the TPM for a subsequent TPM_SHA1Update, TPM_SHA1Complete or TPM_SHA1CompleteExtend command. The capability SHALL open a thread that calculates a SHA-1 digest.

# 13.2 TPM_SHA1Update

***Start of informative comment:***

This capability inputs complete blocks of data into a pending SHA-1 digest. At the end of the process, the digest remains pending.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SHA1Update |
| 4 | 4 | | | UINT32 | numBytes | The number of bytes in hashData. Must be a multiple of 64 bytes. |
| 5 | <> | | | BYTE [ ] | hashData | Bytes to be hashed |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Description**

This command SHALL incorporate complete blocks of data into the digest of an existing SHA-1 thread. Only integral numbers of complete blocks (64 bytes each) can be processed.

# 13.3 TPM_SHA1Complete

***Start of informative comment:***

This capability terminates a pending SHA-1 calculation.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SHA1Complete |
| 4 | 4 | | | UINT32 | hashDataSize | Number of bytes in hashData, MUST be 64 or less |
| 5 | <> | | | BYTE [ ] | hashData | Final bytes to be hashed |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 20 | | | TPM_DIGEST | hashValue | The output of the SHA-1 hash. |

Description

This command SHALL incorporate a partial or complete block of data into the digest of an existing SHA-1 thread, and terminate that thread. hashDataSize MAY have values in the range of 0 through 64, inclusive.

If the SHA-1 thread has received no bytes the TPM SHALL calculate the SHA-1 of the empty buffer.

# 13.4 TPM_SHA1CompleteExtend

***Start of informative comment:***

This capability terminates a pending SHA-1 calculation and EXTENDS the result into a Platform Configuration Register using a SHA-1 hash process.

This command is designed to complete a hash sequence and extend a PCR in memory-less environments.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SHA1CompleteExtend |
| 4 | 4 | | | TPM_PCRINDEX | pcrNum | Index of the PCR to be modified |
| 5 | 4 | | | UINT32 | hashDataSize | Number of bytes in hashData, MUST be 64 or less |
| 6 | <> | | | BYTE [ ] | hashData | Final bytes to be hashed |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 20 | | | TPM_DIGEST | hashValue | The output of the SHA-1 hash. |
| 5 | 20 | | | TPM_PCRVALUE | outDigest | The PCR value after execution of the command. |

**Description**

This command SHALL incorporate a partial or complete block of data into the digest of an existing SHA-1 thread, EXTEND the resultant digest into a PCR, and terminate the thread. hashDataSize MAY have values in the range of 0 through 64, inclusive.

**Actions**

1. Map V1 to TPM_STANY_DATA
2. Map L1 to V1 -> localityModifier
3. If TPM_PERMANENT_DATA -> pcrAttrib [PCRIndex]. pcrExtendLocal[L1] is FALSE return TPM_BAD_LOCALITY

# 13.5 TPM_Sign

***Start of informative comment:***

The Sign command signs data and returns the resulting digital signature

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | Tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Sign. |
| 4 | 4 | | | TPM_KEY_HANDLE | keyHandle | The keyHandle identifier of a loaded key that can perform digital signatures. |
| 5 | 4 | 2s | 4 | UINT32 | areaToSignSize | The size of the areaToSign parameter |
| 6 | <> | 3s | <> | BYTE[] | areaToSign | The value to sign |
| 7 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | 20 | | | TPM_AUTHDATA | privAuth | The authorization digest that authorizes the use of keyHandle. HMAC key: key.usageAuth |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | Tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Sign. |
| 4 | 4 | 3S | 4 | UINT32 | sigSize | The length of the returned digital signature |
| 5 | <> | 4S | <> | BYTE[ ] | Sig | The resulting digital signature. |
| 6 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth |

**Description**

The TPM MUST support all values of areaToSignSize that are legal for the defined signature scheme and key size. The maximum value of areaToSignSize is determined by the defined signature scheme and key size.

In the case of PKCS1v15_SHA1 the areaToSignSize MUST be TPM_DIGEST (the hash size of a sha1 operation - see 8.5.1 TPM_SS_RSASSAPKCS1v15_SHA1). In the case of PKCS1v15_DER the maximum size of areaToSign is k-11 octets, where k is limited by the key size (see 8.5.2 TPM_SS_RSASSAPKCS1v15_DER).

**Actions**

1. The TPM validates the authorization to use the key pointed to by keyHandle.
2. If the areaToSignSize is 0 the TPM returns TPM_BAD_PARAMETER.
3. Validate that keyHandle -> keyUsage is TPM_KEY_SIGN or TPM_KEY_LEGACY, if not return the error code TPM_INVALID_KEYUSAGE
4. The TPM verifies that the signature scheme and key size can properly sign the areaToSign parameter.
5. If signature scheme is PKCSv15_SHA1 then
   a. Validate that areaToSignSize is 20 return TPM_BAD_PARAMETER on error
   b. Set S1 to areaToSign
6. Else if signature scheme is PCKSv15_DER then
   a. Validate that areaToSignSize is at least 11 bytes less than the key size, return TPM_BAD_PARAMETER on error
   b. Set S1 to areaToSign
7. else if signature scheme is PKCSv15_INFO then
   a. Create S1 a TPM_SIGN_INFO structure
   b. Set S1 -> fixed to “SIGN”
   c. Set S1 -> replay to nonceOdd
   d. Set S1 -> dataLen areaToSignSize
   e. Set S1 -> data to areaToSign
   f. Validate that sizeof(S1) is a size valid to be signed by the key pointed to by keyHandle, return TPM_BAD_PARAMETER on error
8. Else return TPM_INVALID_KEYUSAGE
9. The TPM computes the signature, sig, using the key referenced by keyHandle using S1 as the value to sign
10. Return the computed signature in Sig

# 13.6 TPM_GetRandom

***Start of informative comment:***

GetRandom returns the next bytesRequested bytes from the random number generator to the caller.

It is recommended that a TPM implement the RNG in a manner that would allow it to return RNG bytes such that the frequency of bytesRequested being less than the number of bytes available be a infrequent occurrence.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_GetRandom. |
| 4 | 4 | | | UINT32 | bytesRequested | Number of bytes to return |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 4 | | | UINT32 | randomBytesSize | The number of bytes returned |
| 5 | <> | | | BYTE[ ] | randomBytes | The returned bytes |

**Actions**

1. The TPM determines if amount bytesRequested is available from the TPM.
2. Set randomBytesSize to the number of bytes available from the RNG. This number MAY be less than randomBytesSize.
3. Set randomBytes to the next randomBytesSize bytes from the RNG

# 13.7 TPM_StirRandom

***Start of informative comment:***

StirRandom adds entropy to the RNG state.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_StirRandom |
| 4 | 4 | | | UINT32 | dataSize | Number of bytes of input (<256) |
| 5 | <> | | | BYTE[ ] | inData | Data to add entropy to RNG state |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Actions**

The TPM updates the state of the current RNG using the appropriate mixing function.

# 13.8 TPM_CertifyKey

***Start of informative comment:***

The TPM_CERTIFYKEY operation allows a key to certify the public portion of certain storage and signing keys.

A TPM identity key may be used to certify non-migratable keys but is not permitted to certify migratory keys. As such, it allows the TPM to make the statement "this key is held in a TPM-shielded location, and it will never be revealed." For this statement to have veracity, the Challenger must trust the policies used by the Privacy CA that issued the identity and the maintenance policy of the TPM manufacturer.

Signing and legacy keys may be used to certify both migratable and non-migratable keys. Then the usefulness of a certificate depends on the trust in the certifying key by the recipient of the certificate.

The key to be certified must be loaded before TPM_CertifyKey is called.

The determination to use TPM_CERTIFY_INFO or TPM_CERTIFY_INFO2 is based solely on the locality restriction. Any key that does not have any locality restrictions will return a TPM_CERTIFY_INFO structure. This means that both 1.1 and 1.2 version keys with no locality restrictions will return a TPM_CERTIFY_INFO. A 1.2 version key that restricts locality will always return a TPM_CERTIFY_INFO2 structure.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed at TPM_ORD_CertifyKey |
| 4 | 4 | | | TPM_KEY_HANDLE | certHandle | Handle of the key to be used to certify the key. |
| 5 | 4 | | | TPM_KEY_HANDLE | keyHandle | Handle of the key to be certified. |
| 6 | 20 | 2S | 20 | TPM_NONCE | antiReplay | 160 bits of externally supplied data (typically a nonce provided to prevent replay-attacks) |
| 7 | 4 | | | TPM_AUTHHANDLE | certAuthHandle | The authorization handle used for certHandle. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with certAuthHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | 20 | | | TPM_AUTHDATA | certAuth | The authorization digest for inputs and certHandle. HMAC key: certKey.auth. |
| 11 | 4 | | | TPM_AUTHHANDLE | keyAuthHandle | The authorization handle used for the key to be signed. |
| | | 2H2 | 20 | TPM_NONCE | keylastNonceEven | Even nonce previously generated by TPM |
| 12 | 20 | 3H2 | 20 | TPM_NONCE | keynonceOdd | Nonce generated by system associated with keyAuthHandle |
| 13 | 1 | 4H2 | 1 | BOOL | continueKeySession | The continue use flag for the authorization handle |
| 14 | 20 | | | TPM_AUTHDATA | keyAuth | The authorization digest for the inputs and key to be signed. HMAC key: key.usageAuth. |

**Outgoing Operands and Sizes**

| Param | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | Sz | # | Sz | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal TPM_ORD_CertifyKey |
| 4 | <> | 3S | <> | TPM_CERTIFY_INFO | certifyInfo | TPM_CERTIFY_INFO that provides information relative to keyhandle |
| 5 | 4 | 4S | 4 | UINT32 | outDataSize | The used size of the output area for outData |
| 6 | <> | 5S | <> | BYTE[ ] | outData | The signed public key. |
| 7 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with certAuthHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag for cert key session |
| 9 | 20 | | 20 | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters and parentHandle. HMAC key: certKey -> auth. |
| 10 | 20 | 2H2 | 20 | TPM_NONCE | keyNonceEven | Even nonce newly generated by TPM |
| | | 3H2 | 20 | TPM_NONCE | keynonceOdd | Nonce generated by system associated with keyAuthHandle |
| 11 | 1 | 4H2 | 1 | BOOL | continueKeyAuthSession | Continue use flag for target key session |
| 12 | 20 | | | TPM_AUTHDATA | keyAuth | The authorization digest for the target key. HMAC key: key.auth. |

**Actions**

1. The TPM validates that the key pointed to by certHandle has a signature scheme of TPM_SS_RSASSAPKCS1v15_SHA1 or TPM_SS_RSASSAPKCS1v15_INFO
2. The TPM verifies the authorization in certAuthHandle provides authorization to use the key pointed to by certHandle, return TPM_AUTHFAIL on error
3. The TPM verifies the authorization in keyAuthHandle provides authorization to use the key pointed to by keyHandle, return TPM_AUTH2FAIL on error
4. If the key pointed to by certHandle is an identity key (certHandle -> keyUsage is TPM_KEY_IDENTITY)
   a. If keyHandle -> keyInfo -> migratable is TRUE and keyHandle -> keyInfo -> migrateAuthority is FALSE return TPM_MIGRATEFAIL
5. If keyHandle -> pcrInfoSize is not 0
   a. If keyHandle -> keyFlags has pcrIgnoredOnRead set to FALSE
      i. Create a digestAtRelease according to the specified PCR registers and compare to keyHandle -> digestAtRelease and if a mismatch return TPM_WRONGPCRVAL
      ii. If specified validate any locality requests on error TPM_BAD_LOCALITY
   b. Compate h2 with keyHandle -> pcrInfo -> digestAtRelease, on mismatch return TPM_WRONGPCRVALUE
   c. If pcrInfo -> pcrSelection -> SizeOfSelect is 2
      i. This check SHALL only be made when providing backward compatility with 1.1 platforms other TPM implementations SHALL use the TPM_CERTIFY_INFO2 structure
      ii. Create C1 a TPM_CERTIFY_INFO structure

iii. The TPM MUST set c1 -> pcrInfoSize to match the pcrInfoSize from the keyHandle key.
iv. The TPM MUST set c1 -> pcrInfo to match the pcrInfo from the keyHandle key.
v. The TPM MUST set c1 -> digestAtCreation to 20 bytes of 0x00.

d. Else
i. Create C1 a TPM_CERTIFY_INFO2 structure
ii. Set C1 -> pcrInfoSize to keyHandle -> pcrInfoSize
iii. Set C1 -> pcrInfo to keyHandle -> pcrInfo
iv. Set C1 -> pcrInfo -> locality to keyHandle -> locality
v. Set C1 -> migrationAuthoritySize to 0

6. Else
   a. Create C1 a TPM_CERTIFY_INFO structure
   b. Fill in C1 with the information from the key pointed to be keyHandle
   c. The TPM MUST set c1 -> pcrInfoSize to 0
7. Create H1 a TPM_DIGEST to the SHA-1 (keyHandle -> pubKey -> key)
8. Set C1 -> pubKeyDigest to H1
9. The TPM copies the antiReplay parameter to the c1 -> data.
10. The TPM creates m1, a message digest formed by taking the SHA1 of c1.
    a. The TPM then performs a signature using certHandle -> sigScheme. The resulting signed blob is returned in outData.

# 13.9 TPM_CertifyKey2

***Start of informative comment:***

This command provides the ability to certify a Certifiable Migration Key (CMK). This certification requires additional parameters and output then the TPM_CertifyKey. This command always uses the TPM_SIGN_INFO2 structure.

All other aspects of the command are the same as TPM_CertifyKey.

The key to be certified must be loaded before TPM_CertifyKey2 is called.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed at TPM_ORD_CertifyKey2 |
| 4 | 4 | | | TPM_KEY_HANDLE | certHandle | Handle of the key to be used to certify the key. |
| 5 | 4 | | | TPM_KEY_HANDLE | keyHandle | Handle of the key to be certified. |
| 6 | 20 | 2S | 20 | TPM_DIGEST | migrationPubDigest | The digest of the public key of a Migration Authority |
| 7 | 20 | 3S | 20 | TPM_NONCE | antiReplay | 160 bits of externally supplied data (typically a nonce provided to prevent replay-attacks) |
| 8 | 4 | | | TPM_AUTHHANDLE | certAuthHandle | The authorization handle used for certHandle. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 9 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with certAuthHandle |
| 10 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 11 | 20 | | | TPM_AUTHDATA | certAuth | The authorization digest for inputs and certHandle. HMAC key: certKey.auth. |
| 12 | 4 | | | TPM_AUTHHANDLE | keyAuthHandle | The authorization handle used for the key to be signed. |
| | | 2H2 | 20 | TPM_NONCE | keylastNonceEven | Even nonce previously generated by TPM |
| 13 | 20 | 3H2 | 20 | TPM_NONCE | keynonceOdd | Nonce generated by system associated with keyAuthHandle |
| 14 | 1 | 4H2 | 1 | BOOL | continueKeySession | The continue use flag for the authorization handle |
| 15 | 20 | | | TPM_AUTHDATA | keyAuth | The authorization digest for the inputs and key to be signed. HMAC key: key.usageAuth. |

**Outgoing Operands and Sizes**

| Param | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | Sz | # | Sz | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal TPM_ORD_CertifyKey2 |
| 4 | <> | 3S | <> | TPM_CERTIFY_INFO | certifyInfo | TPM_CERTIFY_INFO2 relative to keyHandle |
| 5 | 4 | 4S | 4 | UINT32 | outDataSize | The used size of the output area for outData |
| 6 | <> | 5S | <> | BYTE[ ] | outData | The signed public key. |
| 7 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with certAuthHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag for cert key session |
| 9 | 20 | | 20 | TPM_AUTHDATA | resAuth | Authorization. HMAC key: certKey -> auth. |
| 10 | 20 | 2H2 | 20 | TPM_NONCE | keyNonceEven | Even nonce newly generated by TPM |
| | | 3H2 | 20 | TPM_NONCE | keynonceOdd | Nonce generated by system associated with keyAuthHandle |
| 11 | 1 | 4H2 | 1 | BOOL | continueKeyAuthSession | Continue use flag for target key session |
| 12 | 20 | | | TPM_AUTHDATA | keyAuth | The authorization digest for the target key. HMAC key: key.auth. |

**Actions**

1. The TPM validates that the key pointed to by certHandle has a signature scheme of TPM_SS_RSASSAPKCS1v15_SHA1.
2. The TPM verifies the authorization in certAuthHandle provides authorization to use the key pointed to by certHandle, return TPM_AUTHFAIL on error
3. The TPM verifies the authorization in keyAuthHandle provides authorization to use the key pointed to by keyHandle, return TPM_AUTH2FAIL on error
4. If the key pointed to by certHandle is an identity key (certHandle -> keyUsage is TPM_KEY_IDENTITY), the TPM verifies that the key pointed to by keyHandle is a non-migratory key.
5. The TPM SHALL create a c1 a TPM_CERTIFY_INFO2 structure from the key pointed to by keyHandle
   a. Set C1 -> data to antiReplay
   b. Set C1 -> pubKeyDigest to the SHA (keyHandle -> pubKey -> key)
   c. Copy other keyHandle parameters into C1
6. If certHandle -> payload == TPM_MS_RESTRICT_MIGRATE_AUTH
   a. create thisPubKey, a TPM_PUBKEY structure containing the public key corresponding to certHandle
   b. Verify that [certHandle -> migrationAuth] == SHA1(tpmProof || migrationPubDigest || SHA1(thisPubkey) ) and return error TPM_MA_SOURCE on mismatch
   c. set migrationAuthority = migrationPubDigest
7. Else (if certHandle -> payload != TPM_MS_RESTRICT_MIGRATE_AUTH)
   a. set migrationAuthority = NULL

b. set migrationAuthoritySize =0

8. If keyHandle -> pcrInfoSize is not 0
   a. If keyHandle -> keyFlags has pcrIgnoredOnRead set to FALSE
      i. Create a digestAtRelease according to the specified PCR registers and compare to keyHandle -> digestAtRelease and if a mismatch return TPM_WRONGPCRVAL
      ii. If specified validate any locality requests on error TPM_BAD_LOCALITY
   b. Compate h2 with keyHandle -> pcrInfo -> digestAtRelease, on mismatch return TPM_WRONGPCRVALUE
   c. The TPM MUST set c1 -> pcrInfoSize to match the pcrInfoSize from the keyHandle key.
   d. The TPM MUST set c1 -> pcrInfo to match the pcrInfo from the keyHandle key
9. Else
   a. The TPM MUST set c1 -> pcrInfoSize to 0
10. Set M1 to SHA-1 (C1)
11. Create sig the digital signature of C1 using certHandle -> sigScheme
12. Return sig in outData

# 14. Credential Handling

***Start of informative comment:***

There are two create EK commands. The first matches the 1.1 functionality. The second provides the mechanism to enable revokeEK and provides FIPS 140-2 compatability.

If the TPM is to be built without FIPS compatability then TPM_CreateEndorsementKeyPair is enabled and TPM_RevokeEK must not be available.

If the TPM is built for FIPS compatibility then TPM_CreateEKFips is enabled and TPM_RevokeEK must be available.

***End of informative comment.***

1. If the TPM enables TPM_CreateEndorsementKeyPair
   a. TPM_CreateRevocableEK MUST be disabled
   b. TPM_RevokeTrust MUST be disabled
2. If the TPM enables TPM_CreateRevocableEK
   a. TPM_RevokeTrust MUST be enabled
   b. TPM_CreateEndorsementKeyPair MUST be disabled

# 14.1 TPM_CreateEndorsementKeyPair

***Start of informative comment:***

This command creates the TPM endorsement key. It returns a failure code if an endorsement key already exists.

***End of informative comment.***

This is an optional command

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CreateEndorsementKeyPair |
| 4 | 20 | | | TPM_NONCE | antiReplay | Arbitrary data |
| 5 | <> | | | TPM_KEY_PARMS | keyInfo | Information about key to be created, this includes all algorithm parameters |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | Tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | <> | | | TPM_PUBKEY | pubEndorsementKey | The public endorsement key |
| 5 | 20 | | | TPM_DIGEST | Checksum | Hash of pubEndorsementKey and antiReplay |

**Actions**

1. If an EK already exists, return TPM_DISABLED_CMD
2. Validate the keyInfo parameters for the key description
   a. If the algorithm type is RSA the key length MUST be a minimum of 2048. For interoperability the key length SHOULD be 2048
   b. If the algorithm type is other than RSA the strength provided by the key MUST be comparable to RSA 2048
   c. The other parameters of keyInfo (signatureScheme etc.) are ignored.
3. Create a key pair called the “endorsement key pair” using a TPM-protected capability. The type and size of key are that indicated by keyInfo
4. Create checksum by performing SHA1 on the concatenation of (PUBEK || antiReplay)
5. Store the PRIVEK
6. Set TPM_PERMANENT_FLAGS -> CEKPUsed to TRUE

# 14.2 TPM_CreateRevocableEK

***Start of informative comment:***

This command creates the TPM endorsement key. It returns a failure code if an endorsement key already exists. The TPM vendor may have a separate mechanism to create the EK and "squirt" the value into the TPM.

The input parameters specify whether the EK is capable of being reset, whether the authorization value to reset the EK will be generated by the TPM, and the new authorization value itself if it is not to be generated by the TPM. The output parameter is the new authorization value that must be used when resetting the EK (if it is capable of being reset).

The command TPM_RevokeTrust must be used to reset an EK (if it is capable of being reset).

Owner authorisation is unsuitable for authorizing resetting of an EK: someone with Physical Presence can remove a genuine Owner, install a new Owner, and revoke the EK. The genuine Owner can reinstall, but the platform will have lost its original attestation and may not be trusted by challengers. Therefore if a password is to be used to revoke an EK, it must be a separate password, given to the genuine Owner.

In v1.2 an OEM has extra choices when creating EKs.

a) An OEM could manufacture all of its TPMs with enableRevokeEK==TRUE.

If the OEM has tracked the EKreset passwords for these TPMs, the OEM can give the passwords to customers. The customers can use the passwords as supplied, change the passwords, or clear the EKs and create new EKs with new passwords.

If EKreset passwords are random values, the OEM can discard those values and not give them to customers. There is then a low probability (statistically zero) chance of a local DOS attack to reset the EK by guessing the password. The chance of a remote DOS attack is zero because Physical Presence must also be asserted to use TPM_RevokeTrust.

b) An OEM could manufacture some of its TPMs with enableRevokeEK==FALSE. Then the EK can never be revoked, and the chance of even a local DOS attack on the EK is eliminated.

***End of informative comment.***

This is an optional command

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CreateRevocableEK |
| 4 | 20 | | | TPM_NONCE | antiReplay | Arbitrary data |
| 5 | <> | | | TPM_KEY_PARMS | keyInfo | Information about key to be created, this includes all algorithm parameters |
| 6 | 1 | | | BOOL | generateReset | If TRUE use TPM RNG to generate EKreset. If FALSE use the passed value inputEKreset |
| 7 | 20 | | | TPM_NONCE | inputEKreset | The authorization value to be used with TPM_RevokeTrust if generateReset==FALSE |

**Outgoing Operands and Sizes**

| PARAM | HMAC | Type | Name | Description |
|---|---|---|---|---|

| # | SZ | # | SZ | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | | | TPM_TAG | Tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | <> | | | TPM_PUBKEY | pubEndorsementKey | The public endorsement key |
| 5 | 20 | | | TPM_DIGEST | Checksum | Hash of pubEndorsementKey and antiReplay |
| 6 | 20 | | | TPM_NONCE | outputEKreset | The authorization value to use TPM_RevokeTrust |

**Actions**

1. If an EK already exists, return TPM_DISABLED_CMD
2. Perform the actions of TPM_CreateEndorsementKeyPair, if any errors return with error
3. Set TPM_PERMANENT_DATA -> enableRevokeEK to TRUE
   a. If generateReset is TRUE then
      i. Set TPM_PERMANENT_DATA -> EKreset to the next value from the TPM RNG
   b. Else
      i. Set TPM_PERMANENT_DATA -> EKreset to inputEkreset
4. Return PUBEK, checksum and Ekreset
5. The outputEKreset authorization is sent in the clear. There is no uniqueness on the TPM available to actually perform encryption or use an encrypted channel. The assumption is that this operation is occurring in a controlled environment and sending the value in the clear is acceptable.

# 14.3 TPM_RevokeTrust

***Start of informative comment:***

This command clears the EK and sets the TPM back to a pure default state. The generation of the authorization value occurs during the generation of the EK. It is the responsibility of the EK generator to properly protect and disseminate the RevokeTrust authorization.

This command exists solely to satisfy the FIPS 140 requirement that all keys must be invalidated.

***End of informative comment.***

This is an optional command

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_RevokeTrust |
| 4 | 20 | | | TPM_DIGEST | EKReset | The value that will be matched to EK Reset |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| | | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_RevokeTrust |

**Actions**

1. The TPM MUST validate that TPM_PERMANENT_DATA -> enableRevokeEK is TRUE, return TPM_PERMANENTEK on error
2. The TPM MUST validate that the EKReset matches TPM_PERMANENT_DATA -> EKReset TPM_AUTHFAIL on error.
3. Ensure that physical presence is being asserted
4. Perform the actions of TPM_OwnerClear (excepting the command authentication)
5. Invalidate the EK and any internal state associated with the EK

# 14.4 TPM_ReadPubek

***Start of informative comment:***

Return the endorsement key public portion. This value should have controls placed upon access as it is a privacy sensitive value

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ReadPubek |
| 4 | 20 | | | TPM_NONCE | antiReplay | Arbitrary data |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | <> | | | TPM_PUBKEY | pubEndorsementKey | The public endorsement key |
| 5 | 20 | | | TPM_DIGEST | checksum | Hash of pubEndorsementKey and antiReplay |

**Description**

This command returns the PUBEK.

**Actions**

The TPM_ReadPubek command SHALL

1. If TPM_PERMANENT_FLAGS -> readPubek is FALSE return TPM_DISABLED_CMD.
2. If no EK is present the TPM MUST return TPM_NO_ENDORSEMENT
3. Create checksum by performing SHA1 on the concatenation of (PUBEK || antiReplay).
4. Export the PUBEK and checksum.

# 14.5 TPM_DisablePubekRead

***Start of informative comment:***

The TPM Owner may wish to prevent any entity from reading the PUBEK. This command sets the non-volatile flag so that the TPM_ReadPubek command always returns TPM_DISABLED_CMD.

***End of informative comment.***

### Incoming Operands and Sizes

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_DisablePubekRead |
| 4 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 5 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 7 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

### Outgoing Operands and Sizes

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_DisablePubekRead |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

### Actions

This capability sets the TPM_PERSISTENTFLAGS -> readPubek flag to FALSE.

# 14.6 TPM_OwnerReadInteralPub

***Start of informative comment:***

A TPM Owner authorized command that returns the public portion of the EK or SRK.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_OwnerReadInternalPub |
| 4 | 4 | 2S | 4 | TPM_KEY_HANDLE | keyHandle | Handle for either PUBEK or SRK |
| 5 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 6 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 8 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_OwnerReadInternalPub |
| 4 | <> | 3S | <> | TPM_PUBKEY | publicPortion | The public portion of the requested key |
| 5 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 7 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Actions**

1. Validate the parameters and TPM Owner authorization for this command
2. If keyHandle is TPM_KH_EK
   a. Set publicPortion to PUBEK
3. Else If keyHandle is TPM_KH_SRK
   a. Set publicPortion to the TPM_PUBKEY of the SRK
4. Else return TPM_INVALID_PARAMETER
5. Export the PUBEK

# 15. Identity Creation and Activation

## 15.1 TPM_MakeIdentity

***Start of informative comment:***

Generate a new Attestation Identity Key (AIK)

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes incl. paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_MakeIdentity. |
| 4 | 20 | 2S | 20 | TPM_ENCAUTH | identityAuth | Encrypted usage authorization data for the new identity |
| 5 | 20 | 3S | 20 | TPM_CHOSENID_HASH | labelPrivCADigest | The digest of the identity label and privacy CA chosen for the new TPM identity. (SeeTODOREF for details) |
| 6 | <> | 4S | <> | TPM_KEY | idKeyParams | Structure containing all parameters of new identity key. pubKey.keyLength & idKeyParams.encData are both 0 MAY be TPM_KEY12 |
| 7 | 4 | | | TPM_AUTHHANDLE | srkAuthHandle | The authorization handle used for SRK authorization. |
| | | 2H1 | 20 | TPM_NONCE | srkLastNonceEven | Even nonce previously generated by TPM |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | srknonceOdd | Nonce generated by system associated with srkAuthHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueSrkSession | Ignored |
| 10 | 20 | | | TPM_AUTHDATA | srkAuth | The authorization digest for the inputs and the SRK. HMAC key: srk.usageAuth. |
| 11 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. Session type MUST be OSAP. |
| | | 2H2 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 12 | 20 | 3H2 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 13 | 1 | 4H2 | 1 | BOOL | continueAuthSession | Ignored |
| 14 | 20 | | 20 | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal:TPM_ORD_MakeIdentity. |
| 4 | <> | 3S | <> | TPM_KEY | idKey | The newly created identity key. MAY be TPM_KEY12 |
| 5 | 4 | 4S | 4 | UINT32 | identityBindingSize | The used size of the output area for identityBinding |
| 6 | <> | 5S | <> | BYTE[ ] | identityBinding | Signature of TPM_IDENTITY_CONTENTS using idKey.private. |
| 7 | 20 | 2H2 | 20 | TPM_NONCE | srkNonceEven | Even nonce newly generated by TPM. |
| | | 3H2 | 20 | TPM_NONCE | srknonceOdd | Nonce generated by system associated with srkAuthHandle |
| 8 | 1 | 4H2 | 1 | BOOL | continueSrkSession | Fixed value FALSE |
| 9 | 20 | | | TPM_AUTHDATA | srkAuth | The authorization digest used for the outputs and srkAuth session. HMAC key: srk.usageAuth. |
| 10 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 11 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Fixed value FALSE |
| 12 | 20 | | 20 | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Description**

The public key of the new TPM identity SHALL be identityPubKey. The private key of the new TPM identity SHALL be tpm_signature_key.

**Properties of the new identity**

| Type | Name | Description |
|---|---|---|
| TPM_PUBKEY | identityPubKey | This SHALL be the public key of a previously unused asymmetric key pair. |
| TPM_STORE_ASYMKEY | tpm_signature_key | This SHALL be the private key that forms a pair with identityPubKey and SHALL be extant only in a TPM-shielded location. |

This capability also generates a TPM_KEY containing the tpm_signature_key.

If identityPubKey is stored on a platform it SHALL exist only in storage to which access is controlled and is available to authorized entities.

**Actions**

A Trusted Platform Module that receives a valid TPM_MakeIdentity command SHALL do the following:

1. Validate the idKeyParams parameters for the key description
   a. If the algorithm type is RSA the key length MUST be a minimum of 2048. For interoperability the key length SHOULD be 2048
   b. If the algorithm type is other than RSA the strength provided by the key MUST be comparable to RSA 2048

c. If the TPM is not designed to create a key of the requested type, return the error code TPM_BAD_KEY_PROPERTY

2. Use authHandle to verify that the Owner authorized all TPM_MakeIdentity input parameters.
3. Use srkAuthHandle to verify that the SRK owner authorized all TPM_MakeIdentity input parameters.
4. Verify that idKeyParams -> keyUsage is TPM_KEY_IDENTITY. If it is not, return TPM_INVALID_KEYUSAGE
5. Verify that idKeyParams -> keyFlags -> migratable is FALSE. If it is not, return TPM_INVALID_KEYUSAGE
6. Decrypt the authorization value
   a. Create s1 string by concatenating (ownerAuth -> shared secret || authLastNonceEven)
   b. Create x1 by performing a SHA1 hash of s1
   c. Create A1 by XOR of x1 and identityAuth
7. Set continueAuthSession to FALSE.
8. Determine the structure version
   a. If idKeyParms -> tag is TPM_TAG_KEY12
      i. Set V1 to 2
      ii. Create T1 a TPM_KEY12 structure using idKeyParams as the default values for the structure
   b. If idKeyParms -> ver is 1.1
      i. Set V1 to 1
      ii. Create T1 a TPM_KEY structure using idKeyParams as the default values for the structure
9. Set the digestAtCreation values for pcrInfo
   a. For PCR_INFO_LONG include the locality of the current command
10. Create an asymmetric key pair (identityPubKey and tpm_signature_key) using a TPM-protected capability, in accordance with the algorithm specified in idKeyParams
11. Ensure that the authorization information in A1 is properly stored in the idKey as usageAuth.
12. Attach identityPubKey and tpm_signature_key to idKey
13. Set idKey -> migrationAuth to TPM_PERSISTANT_DATA -> tpmProof
14. Ensure that all TPM_PAYLOAD_TYPE structures identity this key as TPM_PT_ASYM
15. Encrypt the private portion of idKey using the SRK as the parent key
16. Create a TPM_IDENTITY_CONTENTS structure named idContents using labelPrivCADigest and the information from idKey
17. Sign idContents using tpm_signature_key and TPM_SS_RSASSAPKCS1v15_SHA1. Store the result in identityBinding.

# 15.2 TPM_ActivateIdentity

***Start of informative comment:***

The purpose of TPM_ActivateIdentity is to twofold. The first purpose is to obtain assurance that the credential in the TPM_SYM_CA_ATTESTATION is for this TPM. The second purpose is to obtain the session key used to encrypt the TPM_IDENTITY_CREDENTIAL.

This is an extension to the 1.1 functionality of TPM_ActivateIdentity. The blob sent to from the CA can be in the 1.1 format or the 1.2 format. The TPM determines the type from the size or version information in the blob.

TPM_ActivateIdentity checks that the symmetric session key corresponds to a TPM-identity before releasing that session key.

Only the Owner of the TPM has the privilege of activating a TPM identity. The Owner is required to authorize the TPM_ActivateIdentity command. The owner may authorize the command using either the TPM_OIAP or TPM_OSAP authorization protocols.

The creator of the ActivateIdentity package can specify if any PCR values are to be checked before releasing the session key.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes incl. paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ActivateIdentity |
| 4 | 4 | | | TPM_KEY_HANDLE | idKey | Identity key to be activated |
| 5 | 4 | 2S | 4 | UINT32 | blobSize | Size of encrypted blob from CA |
| 6 | <> | 3S | <> | BYTE [ ] | blob | The encrypted ASYM_CA_CONTENTS or TPM_EK_BLOB |
| 7 | 4 | | | TPM_AUTHHANDLE | idKeyAuthHandle | The authorization handle used for ID key authorization. |
| | | 2H1 | 20 | TPM_NONCE | idKeyLastNonceEven | Even nonce previously generated by TPM |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | idKeynonceOdd | Nonce generated by system associated with idKeyAuthHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueIdKeySession | Continue usage flag for idKeyAuthHandle. |
| 10 | 20 | | | TPM_AUTHDATA | idKeyAuth | The authorization digest for the inputs and ID key. HMAC key: idKey.usageAuth. |
| 11 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H2 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 12 | 20 | 3H2 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 13 | 1 | 4H2 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 14 | 20 | | 20 | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner. HMAC key: ownerAuth. |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal:TPM_ORD_ActivateIdentity |
| 4 | <> | 3S | <> | TPM_SYMMETRIC_KEY | symmetricKey | The decrypted symmetric key. |
| 5 | 20 | 2H1 | 20 | TPM_NONCE | idKeyNonceEven | Even nonce newly generated by TPM. |
| | | 3H1 | 20 | TPM_NONCE | idKeynonceOdd | Nonce generated by system associated with idKeyAuthHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueIdKeySession | Continue use flag, TRUE if handle is still active |
| 7 | 20 | | | TPM_AUTHDATA | idKeyAuth | The authorization digest used for the returned parameters and idKeyAuth session. HMAC key: idKey.usageAuth. |
| 8 | 20 | 2H2 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H2 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H2 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 10 | 20 | | 20 | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Description**

The command TPM_ActivateIdentity activates a TPM identity created using the command TPM_MakeIdentity.

The command assumes the availability of the private key associated with the identity. The command will verify the association between the keys during the process.

The command will decrypt the input blob and extract the session key and verify the connection between the public and private keys. The input blob can be in 1.1 or 1.2 format.

**Actions**

A Trusted Platform Module that receives a valid TPM_ActivateIdentity command SHALL do the following:

1. Using the authHandle field, validate the owner's authorization to execute the command and all of the incoming parameters.
2. Using the idKeyAuthHandle, validate the authorization to execute command and all of the incoming parameters
3. Validate that the idKey is the public key of a valid TPM identity by checking that idKey -> keyUsage is TPM_KEY_IDENTITY. Return TPM_BAD_PARAMETER on mismatch
4. Create H1 the digest of the public key in idKey
5. Decrypt blob creating B1 using PRIVEK as the decryption key
6. Determine the type and version of B1
   a. If B1 -> tag is TPM_TAG_EK_BLOB then
      i. B1 is a TPM_EK_BLOB
   b. Else

i. B1 is a TPM_ASYM_CA_CONTENTS. As there is no tag for this structure it is possible for the TPM to make a mistake here but other sections of the structure undergo validation

7. If B1 is a version 1.1 TPM_ASYM_CA_CONTENTS then
   a. Compare H1 to B1 -> idDigest on mismatch return TPM_BAD_PARAMETER
   b. Set K1 to B1 -> sessionKey
8. If B1 is a TPM_EK_BLOB then
   a. Compare H1 to B1 -> idDigest on mismatch return TPM_BAD_PARAMETER
   b. Validate that B1 -> ekType is TPM_EK_TYPE_ACTIVATE, return TPM_BAD_TYPE if not.
   c. Assign A1 as a TPM_EK_TYPE_ACTIVATE structure from B1 -> blob
   d. If A1 -> pcrSelection is not NULL
      i. Compute a composite hash C1 using the PCR selection A1 -> pcrSelection
      ii. Compare C1 to A1 -> compositeHash and return TPM_WRONGPCRVALUE on a mismatch
      iii. If pcrInfo specifies a locality ensure that the appropriate locality has been asserted, return TPM_BAD_LOCALITY on error
   e. Set K1 to A1 -> symmetricKey
9. Return K1

# 16. Integrity Collection and Reporting

***Start of informative comment:***

This section deals with what commands have direct access to the PCR

***End of informative comment.***

1. The TPM SHALL only allow the following commands to alter the value of a PCR
   a. TPM_Extend
   b. TPM_SHA1CompleteExtend
   c. TPM_Startup
   d. TPM_PCRReset

# 16.1 TPM_Extend

***Start of informative comment:***

This adds a new measurement to a PCR

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Extend. |
| 4 | 4 | | | TPM_PCRINDEX | pcrNum | The PCR to be updated. |
| 5 | 20 | | | TPM_DIGEST | inDigest | The 160 bit value representing the event to be recorded. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 20 | | | TPM_PCRVALUE | outDigest | The PCR value after execution of the command. |

**Descriptions**

Add a measurement value to a PCR

**Actions**

4. Map V1 to TPM_STANY_DATA
5. Map L1 to V1 -> localityModifier
6. If TPM_PERMANENT_DATA -> pcrAttrib [PCRIndex]. pcrExtendLocal[L1] is FALSE return TPM_BAD_LOCALITY
7. Create c1 by concatenating (PCRindex TPM_PCRVALUE || inDigest). This takes the current PCR value and concatenates the inDigest parameter.
8. Create h1 by performing a SHA1 digest of c1.
9. Store h1 as the new TPM_PCRVALUE of PCRindex
10. If TPM_PERSISTENT_FLAG -> disable is TRUE
    a. Set outDigest to 20 bytes of 0x00
11. Else
    a. Set outDigest to h1

# 16.2 TPM_PCRRead

***Start of informative comment:***

The TPM_PCRRead operation provides non-cryptographic reporting of the contents of a named PCR.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_PCRRead. |
| 4 | 4 | | | TPM_PCRINDEX | pcrIndex | Index of the PCR to be read |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 20 | | | TPM_PCRVALUE | outDigest | The current contents of the named PCR |

**Actions**

The TPM_PCRRead operation returns the current contents of the named register to the caller.

# 16.3 TPM_Quote

***Start of informative comment:***

The TPM_Quote operation provides cryptographic reporting of PCR values. A loaded key is required for operation. TPM_Quote uses a key to sign a statement that names the current value of a chosen PCR and externally supplied data (which may be a nonce supplied by a Challenger).

The term "ExternalData" is used because an important use of TPM_Quote is to provide a digital signature on arbitrary data, where the signature includes the PCR values of the platform at time of signing. Hence the "ExternalData" is not just for anti-replay purposes, although it is (of course) used for that purpose in an integrity challenge.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Quote. |
| 4 | 4 | | | TPM_KEY_HANDLE | keyHandle | The keyHandle identifier of a loaded key that can sign the PCR values. |
| 5 | 20 | 2S | 20 | TPM_NONCE | extrnalData | 160 bits of externally supplied data (typically a nonce provided by a server to prevent replay-attacks) |
| 6 | <> | 3S | <> | TPM_PCR_SELECTION | targetPCR | The indices of the PCRs that are to be reported. |
| 7 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | 20 | | | TPM_AUTHDATA | privAuth | The authorization digest for inputs and keyHandle. HMAC key: key -> usageAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUT1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Quote. |
| 4 | <> | 3S | <> | TPM_PCR_COMPOSITE | pcrData | A structure containing the same indices as targetPCR, plus the corresponding current PCR values. |
| 5 | 4 | 4S | 4 | UINT32 | sigSize | The used size of the output area for the signature |
| 6 | <> | 5S | <> | BYTE[ ] | sig | The signed data blob. |
| 7 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 9 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: Key -> usageAuth. |

**Actions**

1. The TPM MUST validate the authorization to use the key pointed to by keyHandle.
2. Validate targetPCR
   a. targetPCR is a valid TPM_PCR_SELECTION structure
   b. targetPCR -> pcrSelect is non-zero
   c. On errors return TPM_INVALID_PCR_INFO
3. Create H1 a TPM_PCR_COMPOSITE using the PCR indicated by targetPCR -> pcrSelect
4. Create Q1 a TPM_QUOTE_INFO structure
   a. Set Q1 -> version to 1.1.0.0
   b. Set Q1 -> fixed to “QUOT”
   c. Set Q1 -> digestValue to H1
   d. Set Q1 -> externalData to externalData
5. Sign Q1 using keyHandle as the signature key
6. Return the signature in sig

# 16.4 TPM_PCR_Reset

***Start of informative comment:***

Resets the indicated PCRs. This command uses the locality modifier.

The modifier for a command to indicate locality is a platform specific issue. For the LPC bus the TPM Interface Specification defines the addresses and signals that make up the locality modifier.

A platform specific specification must define the modifier that indicates locality.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed at TPM_ORD_PCR_Reset |
| 4 | <> | | | TPM_PCR_SELECTION | pcrSelection | The PCR's to reset |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Descriptions**

This command resets PCR values back to the default value. The command MUST validate that all PCR registers that are selected are available to be reset before resetting any PCR. This command MUST either reset all selected PCR registers or none of the PCR registers.

**Actions**

1. Validate that pcrSelection is valid
   a. targetPCR is a valid TPM_PCR_SELECTION structure
   b. targetPCR -> pcrSelect is non-zero
   c. On errors return TPM_INVALID_PCR_INFO
2. Map V1 to TPM_STCLEAR_FLAGS
3. For each PCR selected perform the following
4. If pcrAttrib[pcrIndex].pcrReset is FALSE
   a. Return TPM_NOTRESETABLE

5. If pcrAttrib[pcrIndex].pcrResetLocal[ V1 -> localityModifier] is FALSE
   a. Return TPM_NOTLOCAL
6. For each PCR selected perform the following
   a. Reset PCR to default value.

# 17. Authorization Changing

## 17.1 TPM_ChangeAuth

***Start of informative comment:***

The TPM_ChangeAuth command allows the owner of an entity to change the authorization data for the entity.

TPM_ChangeAuth requires the encryption of one parameter ("NewAuth"). For the sake of uniformity with other commands that require the encryption of more than one parameter, the string used for XOR encryption is generated by concatenating the evenNonce (created during the OSAP session) with the session shared secret and then hashing the result.

The parameter list to this command must always include two authorization sessions, regardless of the state of authDataUsage for the respective keys.

***End of informative comment.***

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed at TPM_ORD_ChangeAuth |
| 4 | 4 | | | TPM_KEY_HANDLE | parentHandle | Handle of the parent key to the entity. |
| 5 | 2 | 2 S | 2 | TPM_PROTOCOL_ID | protocolID | The protocol in use. |
| 6 | 20 | 3 S | 20 | TPM_ENCAUTH | newAuth | The encrypted new authorization data for the entity. The encryption key is the shared secret from the OS-AP protocol. |
| 7 | 2 | 4 S | 2 | TPM_ENTITY_TYPE | entityType | The type of entity to be modified |
| 8 | 4 | 5 S | 4 | UINT32 | encDataSize | The size of the encData parameter |
| 9 | <> | 6 S | <> | BYTE[ ] | encData | The encrypted entity that is to be modified. |
| 10 | 4 | | | TPM_AUTHHANDLE | parentAuthHandle | The authorization handle used for the parent key. |
| | | 2 H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 11 | 20 | 3 H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with parentAuthHandle |
| 12 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Ignored, parentAuthHandle is always terminated. |
| 13 | 20 | | | TPM_AUTHDATA | parentAuth | The authorization digest for inputs and parentHandle. HMAC key: parentKey.usageAuth. |
| 14 | 4 | | | TPM_AUTHHANDLE | entityAuthHandle | The authorization handle used for the encrypted entity. The session type MUST be OIAP |
| | | 2 H2 | 20 | TPM_NONCE | entitylastNonceEven | Even nonce previously generated by TPM |
| 15 | 20 | 3 H2 | 20 | TPM_NONCE | entitynonceOdd | Nonce generated by system associated with entityAuthHandle |
| 16 | 1 | 4 H2 | 1 | BOOL | continueEntitySession | Ignored, entityAuthHandle is always terminated. |
| 17 | 20 | | | TPM_AUTHDATA | entityAuth | The authorization digest for the inputs and encrypted entity. HMAC key: entity.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_ChangeAuth |
| 4 | 4 | 3S | 4 | UINT32 | outDataSize | The used size of the output area for outData |
| 5 | <> | 4S | <> | BYTE[ ] | outData | The modified, encrypted entity. |
| 6 | 20 | 2 H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with parentAuthHandle |
| 7 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, fixed value of FALSE |
| 8 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters and parentHandle. HMAC key: parentKey.usageAuth. |
| 9 | 20 | 2 H2 | 20 | TPM_NONCE | entityNonceEven | Even nonce newly generated by TPM to cover entity |
| | | 3 H2 | 20 | TPM_NONCE | entitynonceOdd | Nonce generated by system associated with entityAuthHandle |
| 10 | 1 | 4 H2 | 1 | BOOL | entityContinueAuthSession | Continue use flag, fixed value of FALSE |
| 11 | 20 | | | TPM_AUTHDATA | entityAuth | The authorization digest for the returned parameters and entity. HMAC key: newly changed entity.usageAuth. |

The parentAuthHandle session type MUST be TCPA_PID_OSAP.

**Actions**

1. Verify that entityType is one of TCPA_ET_DATA, TCPA_ET_KEY and return the error TCPA_WRONG_ENTITYTYPE if not.
2. Verify that parentAuthHandle session type is TPM_PID_OSAP return TPM_BAD_MODE on error
3. Verify that entityAuthHandle session type is TPM_PID_OIAP return TPM_BAD_MODE on error
4. The encData field MUST be the encData field from either the TCPA_STORED_DATA or TCPA_KEY structures.
5. Create s1 string by concatenating (parentAuthHandle -> shared secret || authLastNonceEven)
6. Create x1 by performing a SHA1 hash of s1
7. Create decryptAuth by XOR of x1 and newAuth.
8. parentAuthHandle MUST be built using the parent entity's authorization data.
9. The TPM MUST validate the command using the authorization data in the parentAuth parameter
10. After parameter validation the TPM creates b1 by decrypting encData using the key pointed to by parentHandle.
11. The TPM MUST validate that b1 is a valid TPM structure
    a. Decrypt b1
    b. Check the tag, length and authValue for match, return TPM_INVALIDSTRUCTURE on any mismatch
12. The TPM replaces the authorization data for b1 with decryptAuth created above.

13. The TPM encrypts b1 using the appropriate mechanism for the type using the parentKeyHandle to provide the key information.
14. The new blob is returned in outData when appropriate.
15. The TPM MUST enforce the destruction of both the parentAuthHandle and entityAuthHandle sessions.

# 17.2 TPM_ChangeAuthOwner

***Start of informative comment:***

The TPM_ChangeAuthOwner command allows the owner of an entity to change the authorization data for the TPM Owner or the SRK.

This command requires authorization from the current TPM Owner to execute.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ChangeAuthOwner |
| 4 | 2 | 2S | 2 | TPM_PROTOCOL_ID | protocolID | The protocol in use. |
| 5 | 20 | 3S | 20 | TPM_ENCAUTH | newAuth | The encrypted new authorization data for the entity. The encryption key is the shared secret from the OSAP protocol. |
| 6 | 2 | 4S | 2 | TPM_ENTITY_TYPE | entityType | The type of entity to be modified |
| 7 | 4 | | | TPM_AUTHHANDLE | ownerAuthHandle | The authorization handle used for the TPM Owner. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with ownerAuthHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag the TPM ignores this value |
| 10 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and ownerHandle. HMAC key: tpmOwnerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal TPM_ORD_ChangeAuthOwner |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with ownerAuthHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, fixed value of FALSE |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters and ownerHandle. HMAC key: tpmOwnerAuth. This is the new tpmOwnerAuth value if this command changed that value. |

**Descriptions**

1. A TPM MUST support the TPM_PID_ADCP protocol.
2. In this capability, the SRK cannot be accessed as entityType TPM_ET_KEY, since the SRK is not wrapped by a parent key.

**Actions**

1. The ownerAuthHandle session type MUST be TPM_PID_OSAP
2. TPM_PID_ADCP protocol actions
   a. Verify that entityType is either TPM_ET_OWNER or TPM_ET_SRK, and return the error TPM_WRONG_ENTITYTYPE if not.
   b. Create s1 string by concatenating (ownerAuthHandle -> shared secret || authLastNonceEven)
   c. Create x1 by performing a SHA1 hash of s1
   d. Create decryptAuth by XOR of x1 and newAuth.
   e. The TPM MUST enforce the destruction of the ownerAuthHandle session upon completion of this command (successful or unsuccessful). This includes setting continueAuthSession to FALSE
3. Set the authorization data for the indicated entity to decryptAuth
4. If entityType is TPM_ET_OWNER invalidate any OSAP or DSAP sessions connected to the TPM Owner
5. If entityType is TPM_ET_SRK invalidate any OSAP or DSAP sessions connected to the SRK

# 18. Authorization Sessions

## 18.1 TPM_OIAP

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_OIAP. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 4 | | | TPM_AUTHHANDLE | authHandle | Handle that TPM creates that points to the authorization state. |
| 5 | 20 | | | TPM_NONCE | nonceEven | Nonce generated by TPM and associated with session. |

**Actions**

1. The TPM_OIAP command allows the creation of an authorization handle and the tracking of the handle by the TPM. The TPM generates the handle and nonce.
2. The TPM has an internal limit as to the number of handles that may be open at one time, so the request for a new handle may fail if there is insufficient space available.
3. Internally the TPM will do the following:
   a. TPM allocates space to save handle, protocol identification, both nonces and any other information the TPM needs to manage the session.
   b. TPM generates authHandle and nonceEven, returns these to caller
4. On each subsequent use of the OIAP session the TPM MUST generate a new nonceEven value.

### 18.1.1 Actions to validate an OIAP session

***Start of informative comment:***

This section describes the authorization-related actions of a TPM when it receives a command that has been authorized with the OIAP protocol.

Many commands use OIAP authorization. The following description is therefore necessarily abstract.

***End of informative comment.***

**Actions**

The TPM MUST perform the following operations:

1. The TPM MUST verify that the authorization handle (H, say) referenced in the command points to a valid session. If it does not, the TPM returns the error code TPM_INVALID_AUTHHANDLE
2. The TPM SHALL retrieve the latest version of the caller's nonce (nonceOdd) and continueAuthSession flag from the input parameter list, and store it in internal TPM memory with the authSession 'H'.
3. The TPM SHALL retrieve the latest version of the TPM's nonce stored with the authorization session H (authLastNonceEven) computed during the previously executed command.
4. The TPM MUST retrieve the secret authorization data (SecretE, say) of the target entity. The entity and its secret must have been previously loaded into the TPM.
5. The TPM SHALL perform a HMAC calculation using the entity secret data, ordinal, input command parameters and authorization parameters per section TODOREF
6. The TPM SHALL compare HM to the authorization value received in the input parameters. If they are different, the TPM returns the error code TPM_AUTHFAIL if the authorization session is the first session of a command, or TPM_AUTH2FAIL if the authorization session is the second session of a command. Otherwise, the TPM executes the command which (for this example) produces an output that requires authentication.
7. The TPM SHALL generate a nonce (nonceEven).
8. The TPM creates an HMAC digest to authenticate the return code, return values and authorization parameters to the same entity secret per section TODOREF
9. The TPM returns the return code, output parameters, authorization parameters and authorization digest.
10. If the output continueUse flag is FALSE, then the TPM SHALL terminate the session. Future references to H will return an error.

# 18.2 TPM_OSAP

***Start of informative comment:***

The TPM_OSAP command creates the authorization handle, the shared secret and generates nonceEven and nonceEvenOSAP.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_OSAP. |
| 4 | 2 | | | TPM_ENTITY_TYPE | entityType | The type of entity in use |
| 5 | 4 | | | UINT32 | entityValue | The selection value based on entityType, e.g. a keyHandle # |
| 6 | 20 | | | TPM_NONCE | nonceOddOSAP | The nonce generated by the caller associated with the shared secret. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 4 | | | TPM_AUTHHANDLE | authHandle | Handle that TPM creates that points to the authorization state. |
| 5 | 20 | | | TPM_NONCE | nonceEven | Nonce generated by TPM and associated with session. |
| 6 | 20 | | | TPM_NONCE | nonceEvenOSAP | Nonce generated by TPM and associated with shared secret. |

**Description**

1. The TPM_OSAP command allows the creation of an authorization handle and the tracking of the handle by the TPM. The TPM generates the handle, nonceEven and nonceEvenOSAP.
2. The TPM has an internal limit on the number of handles that may be open at one time, so the request for a new handle may fail if there is insufficient space available.
3. The TPM_OSAP allows the binding of an authorization to a specific entity. This allows the caller to continue to send in authorization data for each command but not have to request the information or cache the actual authorization data.
4. When TPM_OSAP is wrapped in an encrypted transport session
   a. For input the only parameter encrypted is nonceOddOSAP
   b. For output the only parameters encrypted are nonceEven and nonceEvenOSAP

**Actions**

1. The TPM creates S1 a storage area that keeps track of the information associated with the authorization.

2. S1 MUST track the following information
   a. Protocol identification
   b. nonceEven
      i. Initialized to the next value from the TPM RNG
   c. nonceEvenOSAP
      i. Initialized to the next value from the TPM RNG
   d. shared secret
   e. Any other internal TPM state the TPM needs to manage the session
3. The TPM calculates the shared secret using an HMAC calculation. The key for the HMAC calculation is the secret authorization data assigned to the key handle identified by entityValue. The input to the HMAC calculation is the concatenation of nonces nonceEvenOSAP and nonceOddOSAP. The output of the HMAC calculation is the shared secret which is saved in the authorization area associated with authHandle
4. If entityType = TPM_ET_KEYHANDLE
   a. The entity to authorize is a key held in the TPM. entityValue contains the keyHandle that holds the key.
   b. If entityValue is TPM_KH_OPERATOR return TPM_BADHANDLE
5. else if entityType = TPM_ET_OWNER
   a. This value indicates that the entity is the TPM owner. entityValue is ignored.
6. else if entityType = TPM_ET_SRK
   a. The entity to authorize is the SRK. entityValue is ignored.
7. else if entityType = TPM_ET_COUNTER
   a. The entity is a montonic counter, entityValue contains the counter handle
8. else if entityType = TPM_ET_NV
   a. The entity is a NV index, entityValue contains the NV index
9. End if
10. On each subsequent use of the OSAP session the TPM MUST generate a new nonce value.
11. The TPM MUST ensure that OSAP shared secret is only available while the OSAP session is valid.
12. The session MUST terminate upon any of the following conditions:
    a. The command that uses the session returns an error
    b. The resource is evicted from the TPM or otherwise invalidated
    c. The session is used in any command for which the shared secret is used to encrypt an input parameter (TPM_ENCAUTH)
    d. The TPM Owner is cleared
    e. TPM_ChangeAuthOwner is executed and this session is attached to the owner authorization
    f. The session explicitly terminated with continueAuth, TPM_Reset or TPM_FlushSpecific

### 18.2.1 Actions to validate an OSAP session

***Start of informative comment:***

This section describes the authorization-related actions of a TPM when it receives a command that has been authorized with the OSAP protocol.

Many commands use OSAP authorization. The following description is therefore necessarily abstract.

***End of informative comment***

**Actions**

1. On reception of a command with ordinal C1 that uses an authorization session, the TPM SHALL perform the following actions:
2. The TPM MUST have been able to retrieve the shared secret (Shared, say) of the target entity when the authorization session was established with TPM_OSAP. The entity and its secret must have been previously loaded into the TPM.
3. The TPM MUST verify that the authorization handle (H, say) referenced in the command points to a valid session. If it does not, the TPM returns the error code TPM_INVALID_AUTHHANDLE.
4. The TPM MUST calculate the HMAC (HM1, say) of the command parameters according to section TODOREF
5. The TPM SHALL compare HM1 to the authorization value received in the command. If they are different, the TPM returns the error code TPM_AUTHFAIL if the authorization session is the first session of a command, or TPM_AUTH2FAIL if the authorization session is the second session of a command., the TPM executes command C1 which produces an output (O, say) that requires authentication and uses a particular return code (RC, say).
6. The TPM SHALL generate the latest version of the even nonce (nonceEven).
7. The TPM MUST calculate the HMAC (HM2) of the return parameters according to section TODOREF
8. The TPM returns HM2 in the parameter list.
9. The TPM SHALL retrieve the continue flag from the received command. If the flag is FALSE, the TPM SHALL terminate the session and destroy the thread associated with handle H.
10. If the shared secret was used to provide confidentiality for data in the received command, the TPM SHALL terminate the session and destroy the thread associated with handle H.
11. Each time that access to an entity (key) is authorized using OSAP, the TPM MUST ensure that the OSAP shared secret is that derived from the entity using TPM_OSAP

# 18.3 TPM_DSAP

***Start of informative comment:***

The TPM_DSAP command creates the authorization handle using a delegated authorization value passed into the command as an encrypted blob or from the internal delegation table. It can be used to start an authorization session for a user key or the owner.

Identically to TPM_OSAP, it generates a shared secret and generates nonceEven and nonceEvenOSAP.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_DSAP. |
| 3 | 2 | | | TPM_ENTITY_TYPE | entityType | The type of delegation information to use |
| 4 | 4 | | | TPM_KEY_HANDLE | KeyHandle | Key for which delegated authority corresponds, or 0 if delegated owner activity. Only relevant if entityValue equals TPM_DELEGATE_USEKEY_BLOB |
| 5 | 4 | | | UINT32 | entityValueSize | The size of entityValue. |
| 5 | <> | | | BYTE [ ] | entityValue | TPM_DELEGATE_USERKEY_BLOB or TPM_DELEGATE_OWNER_BLOB or index MUST not be empty<br>If entityType is TPM_ET_DEL_ROW then entityValue is a TPM_DELEGATE_INDEX |
| 6 | 20 | | | TPM_NONCE | nonceOddDSAP | The nonce generated by the caller associated with the shared secret. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 4 | | | TPM_AUTHHANDLE | authHandle | Handle that TPM creates that points to the authorization state. |
| 5 | 20 | | | TPM_NONCE | nonceEven | Nonce generated by TPM and associated with session. |
| 6 | 20 | | | TPM_NONCE | nonceEvenDSAP | Nonce generated by TPM and associated with shared secret. |

**Description**

1. The TPM_DSAP command allows the creation of an authorization handle and the tracking of the handle by the TPM. The TPM generates the handle, nonceEven and nonceEvenOSAP.
2. The TPM has an internal limit on the number of handles that may be open at one time, so the request for a new handle may fail if there is insufficient space available.
3. The TPM_DSAP allows the binding of a delegated authorization to a specific entity. This allows the caller to continue to send in authorization data for each command but not have to request the information or cache the actual authorization data.

4. On each subsequent use of the DSAP session the TPM MUST generate a new nonce value and check if the ordinal to be executed has delegation to execute. The TPM MUST ensure that the DSAP shared secret is only available while the DSAP session is valid.
5. When TPM_DSAP is wrapped in an encrypted transport session
   a. For input the only parameter encrypted is nonceOddDSAP
   b. For output the only parameters encrypted are nonceEven and nonceEvenDSAP
6. The DSAP session MUST terminate under any of the following conditions
   a. The command that uses the session returns an error
   b. If attached to a key, when the key is evicted from the TPM or otherwise invalidated
   c. The session is used in any command for which the shared secret is used to encrypt an input parameter (TPM_ENCAUTH)
   d. The TPM Owner is cleared
   e. TPM_ChangeAuthOwner is executed and this session is attached to the owner authorization
   f. The session explicitly terminated with continueAuth, TPM_Reset or TPM_FlushSpecific
   g. All DSAP sessions MUST be invalidated when any of the following commands execute:
      i. TPM_Delegate_ManageTables
      ii. TPM_Delegate_IncrementCount
      iii. TPM_Delegate_PreLoad
      iv. TPM_Delegate_LoadBlobOwner

**entityType = TPM_ET_DEL_BLOB**

The entityValue parameter contains a delegation blob structure.

**entityType = TPM_ET_DEL_ROW**

The entityValue parameter contains a row number in the nv Delegation table which should be used for the authorization value.

**Actions**

1. If entityType == TPM_ET_DEL_BLOB
   a. Map entityValue to B1 a TPM_DELEGATE_OWNER_BLOB
   b. Validate that B1 is a valid TPM_DELEGATE_OWNER_BLOB, return TPM_WRONG_ENTITYTYPE on error
   c. Locate B1 -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate row, return TPM_BADINDEX if not found
   d. Set FR to TPM_FAMILY_TABLE.FamTableRow[familyRow]
   e. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
   f. Verify that B1->verificationCount equals FR -> verificationCount.
   g. Validate the integrity of the blob
      i. Copy B1 -> integrityDigest to H2
      ii. Set B1 -> integrityDigest to NULL
      iii. Create H3 the HMAC of B1 using tpmProof as the secret
      iv. Compare H2 to H3 return TPM_AUTHFAIL on mismatch

h. Create S1 a TPM_DELEGATE_SENSITIVE by decryptiong B1 -> sensitiveArea using TPM_DELEGATE_KEY
i. Validate S1 values
   i. S1 -> tag is TPM_TAG_DELEGATE_SENSTIVE
   ii. Return TPM_BAD_DELEGATE on error
j. Set A1 to S1 -> authValue

2. Else if entityType == TPM_ET_DEL_ROW
   a. Verify that entityValue points to a valid row in the delegation table.
   b. Set d1 to the delegation information in the row.
   c. Set a1 to d1->authValue.
   d. Locate D1 -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate that row, return TPM_BADINDEX if not found
   e. Set FR to TPM_FAMILY_TABLE.FamTableRow[familyRow]
   f. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
   g. Verify that d1->verificationCount equals FR -> verificationCount.
3. Else if entityType == TPM_ET_DEL_KEY
   a. Map entityValue to K1 a TPM_DELEGATE_KEY_BLOB
   b. Validate that K1 is a valid TPM_DELEGATE_KEY_BLOB, return TPM_WRONG_ENTITYTYPE on error
   c. Locate K1 -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate that row, return TPM_BADINDEX if not found
   d. Set FR to TPM_FAMILY_TABLE.FamTableRow[familyRow]
   e. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
   f. Verify that K1 -> verificationCount equals FR -> verificationCount.
   g. Validate the integrity of the blob
      i. Copy K1 -> integrityDigest to H2
      ii. Set K1 -> integrityDigest to NULL
      iii. Create H3 the HMAC of K1 using tpmProof as the secret
      iv. Compare H2 to H3 return TPM_AUTHFAIL on mismatch
   h. Validate the K1 -> pubKeyDigest identifies keyHandle, return TPM_KEYNOTFOUND on error
   i. Create S1 a TPM_DELEGATE_SENSITIVE by decryptiong K1 -> sensitiveArea using TPM_DELEGATE_KEY
   j. Validate S1 values
      i. S1 -> tag is TPM_TAG_DELEGATE_SENSTIVE
      ii. Return TPM_BAD_DELEGATE on error
   k. Set A1 to S1 -> authValue
4. Generate a new authorization session handle and reserve space to save protocol identification, shared secret, pcrInfo, both nonces, delegated permission bits and any other information the TPM needs to manage the session.
5. Read two new values from the RNG to generate nonceEven and nonceEvenOSAP.

6. The TPM calculates the shared secret using an HMAC calculation. The key for the HMAC calculation is a1. The input to the HMAC calculation is the concatenation of nonces nonceEvenOSAP and nonceOddOSAP. The output of the HMAC calculation is the shared secret which is saved in the authorization area associated with authHandle.

# 18.4 TPM_SetOwnerPointer

***Start of informative comment:***

This command will set a reference to which secret the TPM will use when executing an owner secret related OIAP or OSAP session.

This command should only be used if legacy code must be enabled for delegation to work.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_ COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes incl. paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Ordinal: TPM_ORD_SetOwnerPointer |
| 4 | 4 | | | TPM_ENTITY_TYPE | entityType | The type of entity in use |
| 5 | 4 | | | UINT32 | entityValue | The selection value based on entityType, |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_ COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation |

**Actions**

1. Map TPM_VOLATILE_DATA to V1
2. If entityType = TPM_ET_DEL_ROW
   a. This value indicates that the entity is a delegate row. entityValue is a delegate index in the delegation table.
   b. Validate that entityValue points to a legal row within the delegate table stored within the TPM. If not ok return TPM_BADINDEX.
   c. The TPM sets V1-> OwnerReference to entityValue
   d. Return TPM_SUCCESS
3. else if entityType = TPM_ET_OWNER
   a. This value indicates that the entity is the TPM owner. entityValue is ignored.
   b. The TPM sets V1-> OwnerReference to TPM_KH_OWNER
   c. Return TPM_SUCCESS
4. Return TPM_FAIL

# 19. Delegation Commands

# 19.1 TPM_Delegate_Manage

***Start of informative comment:***

TPM_Delegate_Manage is the fundamental process for managing the Family tables, including enabling/disabling Delegation for a selected Family. Normally TPM_Delegate_Manage must be executed at least once (to create Family tables for a particular family) before any other type of Delegation command in that family can succeed.

Delegate_Manage is authorized by the TPM Owner if an Owner is installed, because changing a table is a privileged Owner operation. If no Owner is installed, Delegate_Manage requires no privilege to execute. This does not disenfranchise an Owner, since there is no Owner, and simplifies loading of tables during platform manufacture or on first-boot. Burn-out of TPM non-volatile storage by inappropriate use is mitigated by the TPM's normal limits on NV-writes in the absence of an Owner. Tables can be locked after loading, to prevent subsequent tampering, and only unlocked by the Owner, his delegate, or the act of removing the Owner (even if there is no Owner).

TPM_Delegate_Manage command is customized by opcode:

(1) TPM_FAMILY_ENABLE enables/disables use of a family and all the rows of the delegate table belonging to that family,

(2) TPM_FAMILY_ADMIN can be used to prevent further management of the Tables until an Owner is installed, or until the Owner is removed from the TPM. (Note that the Physical Presence command TPM_ForceClear always enables further management, even if TPM_ForceClear is used when no Owner is installed.)

(3) TPM_FAMILY_CREATE creates a new family.

(4) TPM_FAMILY_INVALIDATE invalidates an existing family.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | Tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_Delegate_Manage |
| 4 | 4 | 2S | 4 | TPM_FAMILY_ID | familyID | The familyID that is to be managed |
| 5 | 4 | 3s | 4 | TPM_FAMILY_OPERATION | opFlag | Operation to be performed by this command. |
| 6 | 4 | 4s | 4 | UINT32 | opDataSize | Size in bytes of opData |
| 7 | <> | 5s | <> | BYTE [ ] | opData | Data necessary to implement opFlag |
| 8 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 9 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 10 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 11 | 20 | | | TPM_AUTHDATA | ownerAuth | HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | ordinal: TPM_ORD_Delegate_Manage |
| 4 | 4 | 3S | 4 | UINT32 | retDataSize | Size in bytes of retData |
| 5 | <> | 4S | <> | BYTE [ ] | retData | Returned data |
| 6 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TPM_AUTHDATA | resAuth | HMAC key: ownerAuth. |

**Action**

1. Map PD to TPM_PERMANENT_DATA
2. Map PF to TPM_PERMANENT_FLAGS
3. If TPM Owner is installed
   a. Validate the command and parameters using TPM Owner authorization, return TPM_AUTHFAIL on error
   b. If delegated and opFlag = TPM_FAMILY_CREATE
      i. The TPM MUST ignore familyID and opData -> familyID
   c. Else
      i. Validate the command and parameters using Delegation authorisation, then verify that the current delegation family (familyTable -> FamTableRow -> familyID) == familyID; otherwise return error TPM_DELEGATE_FAMILY
4. Else
   a. If opFlag != TPM_FAMILY_CREATE and (familyTable -> FamTableRow -> flags -> DELEGATE_ADMIN_LOCK) is TRUE return TPM_DELEGATE_LOCK
   b. Validate max NV writes without an owner
      i. Set NV1 to PD -> noOwnerNVWrite
      ii. Increment NV1 by 1
      iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITE
      iv. Set PD -> noOwnerNVWrite to NV1
5. If opFlag == TPM_FAMILY_CREATE
   a. Validate that sufficient space exists within the TPM to store an additional family and map F2 to the newly allocated space.
   b. Validate that opData is a TPM_FAMILY_LABEL
      i. If opDataSize != sizeof(TPM_FAMILY_LABEL) return TPM_BAD_SIZE

c. Map F2 to a TPM_FAMILY_TABLE_ENTRY
   i. Set F2 -> tag to TPM_TAG_FAMILY_TABLE_ENTRY
   ii. Set F2 -> label to opData
d. Increment TPM_PERMANENT_DATA -> TPM_LAST_FAMILYID by 1
e. Set F2 -> familyID = PD -> TPM_LAST_FAMILYID
f. Set F2 -> count = 1
g. Set F2 -> flags -> TPM_FAMFLAG_ENABLE to FALSE
h. Set F2 -> flags -> DELEGATE_ADMIN_LOCK to FALSE
i. Set retDataSize = 4
j. Set retData = F2 -> familyID
k. Return TPM_SUCCESS

6. Locate familyID in the TPM_FAMILY_TABLE and set familyRow to indicate row, return TPM_BADINDEX if not found
   a. If authHandle is of type DSAP then continueAuthSession MUST set to FALSE
   b. Set FR to TPM_FAMILY_TABLE.FamTableRow[familyRow]
   c. Invalidate ALL DSAP sessions, active and saved
7. If opFlag == TPM_FAMILY_ADMIN
   a. Validate that opDataSize == 1, and that opData is a Boolean value.
   b. Set (FR -> flags -> DELEGATE_ADMIN_LOCK) = opData
   c. Set retDataSize = 0
   d. Return TPM_SUCCESS
8. If opflag == TPM_FAMILY_ENABLE
   a. Validate that opDataSize == 1, and that opData is a Boolean value.
   b. Set FR -> flags-> TPM_FAMFLAG_ENABLE = opData
   c. Set retDataSize = 0
   d. Return TPM_SUCCESS
9. If opflag == TPM_FAMILY_INVALIDATE
   a. Invalidate all data associated with familyRow; return TPM_BAD_MODE on failure
   b. Return TPM_SUCCESS

# 19.2 TPM_Delegate_CreateKeyDelegation

***Start of informative comment:***

This command delegates privilege to use a key by creating a blob that can be used by TPM_DSAP.

These blobs CANNOT be used as input data for TPM_Delegate_Delegation because the internal TPM delegate table can store owner delegations only.

(TPM_Delegate_CreateOwnerDelegation must be used to delegate Owner privilege.)

***End of informative comment***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Delegate_CreateKeyDelegation. |
| 4 | 4 | | | TPM_KEY_HANDLE | keyHandle | The keyHandle identifier of a loaded key. |
| 5 | 50 | 2S | 50 | TPM_DELEGATE_PUBLIC | publicInfo | The public information necessary to fill in the blob |
| 6 | 20 | 3S | 20 | TPM_ENCAUTH | delAuth | The encrypted new authorization data for the blob. The encryption key is the shared secret from the OSAP protocol. |
| 7 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | 20 | | | TPM_AUTHDATA | privAuth | The authorization digest that authorizes the use of keyHandle. HMAC key: key.usageAuth |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Delegate_CreateKeyDelegation |
| 4 | 4 | 3S | 4 | UINT32 | blobSize | The length of the returned blob |
| 5 | <> | 4S | <> | TPM_DELEGATE_KEY_BLOB | blob | The partially encrypted delegation information. |
| 6 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth |

**Action**

1. Verify authorization for the command and parameters using privAuth
2. If the key authentication is in fact a delegation, then the TPM SHALL ensure that the delegation bits in PublicInfo do not grant more permissions then currently delegated. Return TPM_BAD_AUTH on failure.
3. Set a1 to be the decryption of delAuth
4. Create h1 the SHA-1 of TPM_STORE_PUBKEY structure of the key pointed to by keyHandle
5. Locate publicInfo -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate row, return TPM_BADINDEX if not found
6. Set FR to TPM_FAMILY_TABLE.FamTableRow[familyRow]
7. If FR -> flags -> TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
8. Create M1 a TPM_DELEGATE_SENSITIVE structure
   a. Set M1 -> tag to TPM_TAG_DELEGATE_SENSITIVE
   b. Set M1 -> authValue to a1
   c. The TPM MAY add additional information of a sensitive nature relative to the delegation
9. Create M2 the encryption of M1 using TPM_DELEGATE_KEY
10. Create P1 a TPM_DELEGATE_KEY_BLOB
    a. Set P1 -> tag to TPM_TAG_DELG_KEY_BLOB
    b. Set P1 -> pubKeyDigest to H1
    c. Set P1 -> pub to PublicInfo
    d. Set P1 -> integrityDigest to NULL
    e. The TPM sets additionalArea and additionalAreaSize appropriate for this TPM. The information MAY include symmetric IV, symmetric mode of encryption and other data that allows the TPM to process the blob in the future.
    f. Set P1 -> sensitiveSize to the size of M2
    g. Set P1 -> sensitiveArea to M2
11. Calculate H2 the HMAC of P1 using tpmProof as the secret
12. Set P1 -> integrityDigest to H2
13. Ignore continueAuthSession on input set continueAuthSession to FALSE on output
14. Return P1

# 19.3 TPM_Delegate_CreateOwnerDelegation

***Start of informative comment:***

TPM_Delegate_CreateOwnerDelegation delegates the Owner's privilege to use a set of command ordinals, by creating a blob. Such blobs can be used as input data for TPM_DSAP or TPM_Delegate_LoadOwnerDelegation.

TPM_Delegate_CreateOwnerDelegation includes the ability to void all existing delegations (by incrementing the verification count) before creating the new delegation. This ensures that the new delegation will be the only delegation that can operate at Owner privilege in this family. This new delegation could be used to enable a security monitor (a local separate entity, or remote separate entity, or local host entity) to reinitialize a family and perhaps perform external verification of delegation settings. Normally the ordinals for a delegated security monitor would include TPM_Delegate_CreateKeyDelegation (this command) in order to permit the monitor to create further delegations, and TPM_Delegate_UpdateVerification to reactivate some previously voided delegations.

If the verification count is incremented and the new delegation does not delegate any privileges (to any ordinals) at all, or uses an authorisation value that is then discarded, this family's delegations are all void and delegation must be managed using actual Owner authorisation.

(TPM_Delegate_CreateKeyDelegation must be used to delegate privilege to use a key.)

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | TPM_ORD_Delegate_CreateOwnerDelegation. |
| 4 | 1 | 2S | 1 | BOOL | increment | Flag dictates whether verificationCount will be incremented |
| 5 | 50 | 3S | 50 | TPM_DELEGATE_PUBLIC | publicInfo | The public parameters for the blob |
| 6 | 20 | 4S | 20 | TPM_ENCAUTH | delAuth | The encrypted new authorization data for the blob. The encryption key is the shared secret from the OSAP protocol. |
| 7 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle TPM Ownerauthorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | 20 | | | TPM_AUTHDATA | privAuth | The authorization digest. HMAC key:TPM Owner Auth |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | TPM_ORD_Delegate_CreateOwnerDelegation |
| 4 | 4 | 3S | 4 | UINT32 | blobSize | The length of the returned blob |

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 | <> | 4S | <> | TPM_DELEGATE_USER_BLOB | blob | The partially encrypted delegation information. |
| 6 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: TPM Owner Auth |

Action

1. The TPM SHALL authenticate the command using TPM Owner authentication. Return TPM_AUTHFAIL on failure.
2. If the TPM Owner authentication is in fact a delegation, then the TPM SHALL validate the command and parameters using Delegation authorisation, then
   a. Validate that delegation -> familyID equals publicInfo -> familyID return TPM_DELEGATE_FAMILY or error
   b. If TPM_FAMILY_TABLE.FamTableRow[ delegation -> familyID] -> flags -> TPM_FAMFLAG_ENABLED is FALSE, return error TPM_DISABLED_CMD.
   c. Verify that the delegation bits in publicInfo do not grant more permissions then currently delegated. Otherwise return error TPM_BAD_AUTH.
3. Locate publicInfo -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate the row return TPM_BADINDEX if not found
   a. Set FR to TPM_FAMILY_TABLE.FamTableRow[familyRow]
4. If increment == TRUE
   a. Increment FR -> count
5. Create a1 the decrypted delAuth parameter
6. Create M1 a TPM_DELEGATE_SENSITIVE structure
   a. Set M1 -> tag to TPM_TAG_DELEGATE_SENSITIVE
   b. Set M1 -> authValue to a1
   c. Set other M1 fields as determined by the TPM vendor
7. Create M2 the encryption of M1 using TPM_DELEGATE_KEY
8. Create B1 a TPM_DELEGATE_OWNER_BLOB
   a. If delegated, verify that (B1 -> pub -> familyID) is the current familyID and return error TPM_DELEGATE_FAMILY otherwise
   b. Set B1 -> tag to TPM_TAG_DELG_OWNER_BLOB
   c. Set B1 -> pub to publicInfo
   d. Set B1 -> sensitiveSize to the size of M2
   e. Set B1 -> sensitiveArea to M2
   f. Set B1 -> integrityDigest to NULL
   g. Set B1 -> verificationCount to FR -> count

9. The TPM sets additionalArea and additionalAreaSize appropriate for this TPM. The information MAY include symmetric IV, symmetric mode of encryption and other data that allows the TPM to process the blob in the future.
10. Create H1 the HMAC of B1 using tpmProof as the secret
11. Set B1 -> integrityDigest to H1
12. Ignore continueAuthSession on input set continueAuthSession to FALSE on output
13. Return B1 as blob

# 19.4 TPM_Delegate_LoadOwnerDelegation

***Start of informative comment:***

This command loads a delegate table row blob into a non-volatile delegate table row. Delegate_LoadOwnerDelegation can be used during manufacturing or on first boot (when no Owner is installed), or after an Owner is installed. If an Owner is installed, Delegate_LoadOwnerDelegation requires Owner authorisation, and sensitive information must be encrypted.

Burn-out of TPM non-volatile storage by inappropriate use is mitigated by the TPM's normal limits on NV-writes in the absence of an Owner. Tables can be locked after loading using TPM_Delegate_Manage, to prevent subsequent tampering.

A management system outside the TPM is expected to manage the delegate table rows stored on the TPM, and can overwrite any previously stored data.

This command cannot be used to load key delegation blobs into the TPM

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes incl. paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Ordinal: TPM_ORD_Delegate_LoadDelegationOwner |
| 4 | 4 | 3S | 4 | TPM_DELEGATE_INDEX | index | The index of the delegate row to be written |
| 5 | 4 | 4S | 4 | UINT32 | blobSize | The size of the delegate blob |
| 6 | <> | 5S | <> | TPM_DELEGATE_OWNER_BLOB | blob | Delegation information, including encrypted portions as appropriate |
| 7 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle TPM Ownerauthorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest. HMAC key:TPM Owner Auth |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | TPM_ORD_Delegate_LoadDelegationOwner |
| 5 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 7 | 20 | | | TPM_AUTHDATA | resAuth | Authorization HMAC key: ownerAuth. |

**Actions**

1. If TPM Owner is installed
   a. Validate the command and parameters using TPM Owner authorization, return TPM_AUTHFAIL on error
   b. If delegated, verify that blob -> familyID matches delegate -> familyID on error return TPM_DELEGATE_FAMILY
2. Else
   a. If blob -> familyID -> flags -> DELEGATE_ADMIN_LOCK) is TRUE return TPM_DELEGATE_LOCK
   b. Validate max NV writes without an owner
      i. Set NV1 to PD -> noOwnerNVWrite
      ii. Increment NV1 by 1
      iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITE
      iv. Set PD -> noOwnerNVWrite to NV1
3. Map delegate to D1 a TPM_DELEGATE_OWNER_BLOB
4. Locate D1 -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate row, return TPM_BADINDEX if not found
5. Set FR to TPM_FAMILY_TABLE.FamTableRow[familyRow]
6. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
7. Validate that D1 -> tag == TPM_TAG_DELG_OWNER_BLOB
8. If TPM Owner is installed, validate the integrity of the blob
   a. Copy D1 -> integrityDigest to H2
   b. Set D1 -> integrityDigest to NULL
   c. Create H3 the HMAC of D1 using tpmProof as the secret
   d. Compare H2 to H3 return TPM_AUTHFAIL on mismatch
9. If TPM Owner is installed, create S1 a TPM_DELEGATE_SENSITIVE area by decrypting D1 -> sensitiveArea using TPM_DELEGATE_KEY. Otherwise set S1 = D1 -> sensitiveArea
10. Validate S1
    a. S1 -> tag is TPM_TAG_DELEGATE_SENSITIVE
    b. Return TPM_BAD_DELEGATE on error
11. Validate that index is a valid value for delegateTable, return TPM_BADINDEX on error
12. Copy pcrInfo, authValue, rowLabel, familyID, verificationCount and permissions from delegate into delegateTable[index].
    a. Invalidate ALL DSAP sessions
13. If authHandle is of type DSAP then continueAuthSession MUST set to FALSE
14. Return TPM_SUCCESS

# 19.5 TPM_Delegate_ReadTable

***Start of informative comment:***

This command is used to read from the TPM the public contents of the family and delegate tables that are stored on the TPM. Such data is required during external verification of tables.

There are no restrictions on the execution of this command; anyone can read this information regardless of the state of the PCRs, regardless of whether they know any specific authorization value and regardless of whether or not the enable and admin bits are set one way or the other.

***End of informative comment*.**

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | Ordinal | Command ordinal TPM_ORD_Delegate_ReadTable |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation |
| 4 | 4 | | | UINT32 | familyTableSize | Size in bytes of familyTable |
| 5 | <> | | | BYTE [ ] | familyTable | Array of TPM_FAMILY_TABLE_ENTRY elements |
| 6 | 4 | | | UINT32 | delegateTableSize | Size in bytes of delegateTable |
| 7 | <> | | | TPM_DELEGATE_PUBLIC[] | delegateTable | Array of TPM_DELEGATE_TABLE_PUBLIC elements |

Actions

1. The TPM MUST ensure that this command is not operational when the TPM is disabled or deactivated.
2. Set familyTableSize to the number of valid families on the TPM times sizeof(TPM_FAMILY_TABLE_ELEMENT).
3. Copy the valid entries in the internal family table to the output array familyTable
4. Set delegateTableSize to the number of valid delegate table entries on the TPM times sizeof(TPM_DELEGATE_PUBLIC).
5. For each valid entry
   a. Write the TPM_DELEGATE_INDEX to the output array
   b. Copy the TPM_DELEGATE_PUBLIC to the output array
6. Return TPM_SUCCESS

# 19.6 TPM_Delegate_UpdateVerification

***Start of informative comment:***

UpdateVerification sets the verificationCount in an entity (a blob or a delegation row) to the current family value, in order that the delegations represented by that entity will continue to be accepted by the TPM.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | Tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_Delegate_Update |
| 4 | 4 | 2S | 4 | UINT32 | inputSize | The size of inputData |
| 5 | <> | 3S | <> | BYTE | inputData | TPM_DELEGATE_KEY_BLOB or TPM_DELEGATE_OWNER_BLOB or table index |
| 6 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 7 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 9 | 20 | | | TPM_AUTHDATA | ownerAuth | Authorization HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_Delegate_UpdateVerification |
| 4 | 4 | 3S | 4 | UINT32 | outputSize | The size of the output |
| 5 | <> | 4S | <> | BYTE | outputData | TPM_DELEGATE_KEY_BLOB or TPM_DELEGATE_OWNER_BLOB |
| 6 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Actions**

1. Verify the TPM Owner authorizes the command and parameters, on error return TPM_AUTHFAIL
2. Determine the type of inputData (TPM_DELEGATE_TABLE_ROW or TPM_TAG_DELG_OWNER_BLOB or TPM_DELEGATE_KEY_BLOB) and map D1 to that structure

3. Locate (D1 -> pub -> familyID) in the TPM_FAMILY_TABLE and set familyRow to indicate row, return TPM_BADINDEX if not found
4. Set FR to TPM_FAMILY_TABLE.FamTableRow[familyRow]
5. If delegated, verify that the current delegation family (FR -> familyID) == (D1 -> pub -> familyID); otherwise return error TPM_DELEGATE_FAMILY
6. If delegated AND (FR -> flags TPM_FAMFLAG_ENABLED) is FALSE, return TPM_DISABLED_CMD
7. Set D1 -> verificationCount to FR -> verificationCount
8. If D1 is a blob recreate the blob and return it

# 19.7 TPM_Delegate_VerifyDelegation

***Start of informative comment:***

VerifyDelegation loads a delegate blob and returns success or failure, depending on whether the blob is currently valid.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, TPM_Delegate_SignBlob |
| 4 | 4 | | | UINT32 | delegateSize | The length of the delegated information blob |
| 5 | <> | | | BYTE[ ] | delegation | TPM_DELEGATE_KEY_BLOB or TPM_DELEGATE_OWNER_BLOB |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Actions**

1. Determine the type of blob, If delegation -> tag is equal to TPM_TAG_DELG_OWNER_BLOB then
   a. Map D1 a TPM_DELEGATE_BLOB_OWNER to delegation
2. Else
   a. Map D1 a TPM_DELEGATE_KEY_BLOB to delegation
3. Locate D1 -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate row, return TPM_BADINDEX if not found
4. Set FR to TPM_FAMILY_TABLE.FamTableRow[familyRow]
5. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
6. Validate that D1 -> verificationCount matches FR -> verificationCount, on mismatch return TPM_FAMILYCOUNT
7. Validate the integrity of D1
   a. Copy D1 -> integrityDigest to H2
   b. Set D1 -> integrityDigest to NULL
   c. Create H3 the HMAC of D1 using tpmProof as the secret
   d. Compare H2 to H3 return TPM_AUTHFAIL on mismatch
8. Validate S1 values
   a. S1 -> tag is TPM_TAG_DELEGATE_SENSTIVE

b. Return TPM_BAD_DELEGATE on error

9. Return TPM_SUCCESS

# 20. Non-volatile Storage

***Start of informative comment:***

This section handles the allocation and use of the TPM non-volatile storage.

***End of informative comment.***

# 20.1 TPM_NV_DefineSpace

***Start of informative comment:***

This establishes the space necessary for the indicated index. The definition will include the access requirements for writing and reading the area.

The space definition size does not include the area needed to manage the space.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Ordinal, TPM_ORD_NV_DefineSpace |
| 4 | 66 | 2S | 66 | TPM_NV_DATA_PUBLIC | pubInfo | The public parameters of the NV area |
| 5 | 20 | 3S | 20 | TPM_ENC_AUTH | encAuth | The encrypted authorization, only valid if the attributes require subsequent authorization |
| 6 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for ownerAuth |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 7 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 9 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest HMAC key: TPM Owner auth |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | ordinal, TPM_ORD_NV_DefineSpace |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest HMAC key: TPM Owner auth |

**Actions**

1. If tag = TPM_TAG_RQU_AUTH1_COMMAND then
   a. The TPM MUST validate the command and parameters using the TPM Owner authorization and ownerAuth, on error return TPM_AUTHFAIL
   b. authHandle session type MUST be OSAP

c. Create X1 the SHA-1 result of the concatenation of (authHandle session secret || authLastNonceEven)
   i. Create A1 the XOR of pubInfo -> encAuth and X1

2. else
   a. Validate the assertion of physical presence. Return TPM_BAD_PRESENCE on error.
   b. If TPM Owner is present then return TPM_OWNERSET.
   c. If pubInfo -> dataSize is 0 then return TPM_BAD_DATASIZE. Setting the size to 0 represents an attempt to delete the value without TPM Owner authorization.
   d. Validate max NV writes without an owner
      i. Set NV1 to TPM_PERMENANT_DATA -> noOwnerNVWrite
      ii. Increment NV1 by 1
      iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITE
      iv. Set TPM_PERMENANT_DATA -> noOwnerNVWrite to NV1
   e. Set A1 to encAuth. There is no nonces or authorization to create the encryption string, hence the authorization value is passed in the clear
3. If pubInfo -> nvIndex is listed in reserved TPM_NV_INDEX table in Part 2 the structure document then return TPM_BADINDEX
4. If pubInfo -> nvIndex points to a valid previously defined storage area then
   a. Map D1 a TPM_NV_DATA_SENSITIVE to the storage area
   b. If D1 -> attributes specifies TPM_NV_PER_GLOBALLOCK then
      i. If TPM_STCLEAR_FLAGS -> bGlobalLock is TRUE then return TPM_AREA_LOCKED
   c. If D1 -> attributes specifies TPM_NV_PER_WRITE_STCLEAR
      i. If D1 -> pubInfo -> bWriteSTClear is TRUE then return TPM_AREA_LOCKED
   d. Set data area currently pointed to by D1 to 0xFF
   e. If pubInfo -> dataSize is 0 then return TPM_SUCCESS
5. Parse pubInfo -> pcrInfoRead
   a. Validate pcrInfoRead structure tag on error return TPM_INVALID_STRUCTURE
   b. If pcrInfoRead -> pcrInfoSize is 0
      i. Set readPCR to FALSE
   c. Else
      i. Set readPCR to TRUE
   d. If pcrInfoRead -> localityAtRelease disallows some localities
      i. Set readLocalities to TRUE
   e. Else
      i. Set readLocalities to FALSE
6. Parse pcrInfoWrite
   a. Validate pcrInfoWrite structure tag on error return TPM_INVALID_STRUCTURE
   b. If pcrInfoWrite -> pcrInfoSize is 0

i. Set writePCR to FALSE

c. Else

i. Set writePCR to TRUE

d. If pcrInfoWrite -> localityAtRelease disallows some localities

i. Set writeLocalities to TRUE

e. Else

i. Set writeLocalities to FALSE

7. Validate that the attributes are consistent
   a. If TPM_NV_PER_OWNERWRITE is TRUE and TPM_NV_PER_AUTHWRITE is TRUE return TPM_AUTH_CONFLICT
   b. If TPM_NV_PER_OWNERREAD is TRUE and TPM_NV_PER_AUTHREAD is TRUE return TPM_AUTH_CONFLICT
   c. If TPM_NV_PER_PCRREAD is TRUE and readPCR is FALSE return TPM_BAD_ATTRIBUTES
   d. If TPM_NV_PER_PCRWRITE is TRUE and writePCR is FALSE return TPM_BAD_ATTRIBUTES
   e. If TPM_NV_PER_OWNERWRITE and TPM_NV_PER_AUTHWRITE and TPM_NV_PER_WRITEDEFINE and TPM_NV_PER_PPWRITE and TPM_NV_PER_PCRWRITE and writeLocalities are all FALSE
      i. Return TPM_PER_NOWRITE
   f. If TPM_NV_PER_PCRREAD then
      i. If pcrInfoRead is NULL return TPM_BAD_ATTRIBUTES
   g. If TPM_NV_PER_PCRWRITE then
      i. Validate pcrInfoWrite structure tag on error return TPM_INVALID_STRUCTURE
      ii. If pcrInfoWrite is NULL return TPM_BAD_ATTRIBUTES
   h. Validate nvIndex
      i. Make sure that the index is applicable for this TPM return TPM_BAD_INDEX on error
   i. If dataSize is 0 return TPM_BAD_SIZE
8. Create D1 a TPM_NV_DATA_SENSITIVE structure
9. Validate that sufficient NV is available to store the data
   a. return TPM_NOSPACE if pubInfo -> dataSize is not available in the TPM
10. Ensure that the TPM reserves the space for dataSize
    a. Set all bytes in the newly defined area to 0xFF
11. Set D1 -> pubInfo to pubInfo
12. Set D1 -> authValue to A1
13. Set bReadSTClear = FALSE;
14. Set bWriteSTClear = FALSE;
15. Set bWriteDefine = FALSE;
16. Ignore continueAuthSession on input and set to FALSE on output
17. Return TPM_SUCCESS

# 20.2 TPM_NV_WriteValue

***Start of informative comment:***

This command writes the value to a defined area. The write can be TPM Owner authorized or unauthorized and protected by other attributes and will work when no TPM Owner is present.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Ordinal, TPM_ORD_NV_WRITEVALUE |
| 4 | 4 | 2S | 4 | TPM_NV_INDEX | nvIndex | The index of the area to set |
| 5 | 4 | 3S | 4 | UINT32 | offset | The offset into the NV Area |
| 6 | 4 | 4S | 4 | UINT32 | dataSize | The size of the data parameter |
| 7 | <> | 5S | <> | BYTE | data | The data to set the area to |
| 8 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for TPM Owner |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 9 | 20 | 3H1 | 20 | TPM_NONCE | authNonceOdd | Nonce generated by caller |
| 10 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 11 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest HMAC key: TPM Owner auth |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | ordinal, TPM_ORD_NV_WRITEVALUE |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | authNonceOdd | Nonce generated by caller |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest HMAC key: TPM Owner auth |

**Actions**

1. If nvIndex = 0 then
   a. Set TPM_STCLEAR_FLAGS -> bGlobalLock to TRUE
   b. return TPM_SUCCESS

2. Locate and set D1 to the TPM_NV_DATA_AREA that corresponds to nvIndex, return TPM_BAD_INDEX on error
3. If tag = TPM_TAG_RQU_AUTH1_COMMAND then
   a. If D1 -> TPM_NV_OWNERWRITE is FALSE return TPM_AUTH_CONFLICT
   b. Validate command and parameters using ownerAuth HMAC with TPM Owner authorization as the secret, return TPM_AUTHFAIL on error
4. Else
   a. If D1 -> TPM_NV_PER_OWNERWRITE is TRUE return TPM_AUTH_CONFLICT
   b. If no TPM Owner validate max NV writes without an owner
      i. Set NV1 to TPM_PERMENANT_DATA -> noOwnerNVWrite
      ii. Increment NV1 by 1
      iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITE
      iv. Set TPM_PERMENANT_DATA -> noOwnerNVWrite to NV1
5. Check that D1 -> pcrInfoWrite -> localityAtRelease for TPM_STANY_DATA -> localityModifier is TRUE
   a. For example if TPM_STANY_DATA -> localityModifier was 2 then D1 -> pcrInfo -> localityAtRelease -> TPM_LOC_TWO would have to be TRUE
   b. On error return TPM_BAD_LOCALITY
6. If D1 -> attributes specifies TPM_NV_PER_PPWRITE then validate physical presence is asserted if not return TPM_BAD_PRESENCE
7. If D1 -> attributes specifies TPM_NV_PER_WRITEDEFINE
   a. If D1 -> bWriteDefine is TRUE return TPM_AREA_LOCKED
8. If D1 -> attributes specifies TPM_NV_PER_GLOBALLOCK
   a. If TPM_STCLEAR_DATA -> bGlobalLockis TRUE return TPM_AREA_LOCKED
9. If D1 -> attributes specifies TPM_NV_PER_WRITE_STCLEAR
   a. If D1 ->bWriteSTClear is TRUE return TPM_AREA_LOCKED
10. If D1 -> attributes specifies TPM_NV_PER_PCRWRITE then
    a. Create P1 a composite hash of the PCR specified by D1 -> pcrSelectionWrite
    b. Compate P1 to digestAtRelease return TPM_WRONGPCRVALUE on mismatch
11. If dataSize = 0 then
    a. Set D1 -> bWriteSTClear to TRUE
    b. Set D1 -> bWriteDefine to TRUE
12. Else
    a. Set S1 to offset + dataSize
    b. If S1 > D1 -> dataSize return TPM_NOSPACE
    c. If D1 -> attributes specifies TPM_PER_WRITEALL
       i. If dataSize != D1 -> dataSize return TPM_NOT_FULLWRITE
    d. Write the new value into the NV storage area

13. Set D1 -> bReadSTClear to FALSE
14. Return TPM_SUCCESS

# 20.3 TPM_NV_WriteValueAuth

***Start of informative comment:***

This command writes to a previously defined area. The area must require authorization to write. This command is for using when authorization other than the owner authorization is to be used. Otherwise, you should use TPM_NV_WriteValue

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | Tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Ordinal, TPM_ORD_NV_WRITEVALUEAUTH |
| 4 | 4 | 2S | 4 | TPM_NV_INDEX | nvIndex | The index of the area to set |
| 5 | 4 | 3S | 4 | UINT32 | offset | The offset into the chunk |
| 6 | 4 | 4S | 4 | UINT32 | dataSize | The size of the data area |
| 7 | <> | 5S | <> | BYTE | data | The data to set the area to |
| 8 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for NV element authorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 9 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 10 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 11 | 20 | | | TPM_AUTHDATA | authValue | HMAC key: NV element auth value |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | ordinal, TPM_ORD_NV_WRITEVALUEAUTH |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | NonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | authValue | HMAC key: NV element auth value |

**Actions**

1. Locate and set D1 to the TPM_NV_DATA_AREA that corresponds to nvIndex, return TPM_BAD_INDEX on error
2. If D1 -> attributes does not specify TPM_NV_PER_AUTHWRITE then return TPM_AUTH_CONFLICT

3. Validate authValue using D1 -> authValue, return TPM_AUTHFAIL on error
4. Check that D1 -> pcrInfoWrite -> localityAtRelease for TPM_STANY_DATA -> localityModifier is TRUE
   a. For example if TPM_STANY_DATA -> localityModifier was 2 then D1 -> pcrInfo -> localityAtRelease -> TPM_LOC_TWO would have to be TRUE
   b. On error return TPM_BAD_LOCALITY
5. If D1 -> attributes specifies TPM_NV_PER_PPWRITE then validate physical presence is asserted if not return TPM_BAD_PRESENCE
6. If D1 -> attributes specifies TPM_NV_PER_PCRWRITE then
   a. Create P1 a composite hash of the PCR specified by D1 -> pcrSelectionWrite
   b. Compate P1 to digestAtRelease return TPM_WRONGPCRVALUE on mismatch
7. If D1 -> attributes specifies TPM_NV_PER_WRITEDEFINE
   a. If D1 -> bWriteDefine is TRUE return TPM_AREA_LOCKED
8. If D1 -> attributes specifies TPM_NV_PER_GLOBALLOCK
   a. If TPM_STCLEAR_FLAGS -> bGlobalLock is TRUE return TPM_AREA_LOCKED
9. If D1 -> attributes specifies TPM_NV_PER_WRITE_STCLEAR
   a. If D1 -> bWriteSTClear is TRUE return TPM_AREA_LOCKED
10. If dataSize = 0 then
    a. Set D1 -> bWriteSTClear to TRUE
    b. Set D1 -> bWriteDefine to TRUE
11. Else
    a. Set S1 to offset + dataSize
    b. If S1 > D1 -> dataSize return TPM_NOSPACE
    c. If D1 -> attributes specifies TPM_PER_WRITEALL
       i. If dataSize != D1 -> dataSize return TPM_NOT_FULLWRITE
    d. Write the new value into the NV storage area
12. Set D1 -> bReadSTClear to FALSE
13. Return TPM_SUCCESS

# 20.4 TPM_NV_ReadValue

***Start of informative comment:***

Read a value from the NV store. This command uses optional owner authorization.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | Tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Ordinal, TPM_ORD_NV_READVALUE |
| 4 | 4 | 2S | 4 | TPM_NV_INDEX | nvIndex | The index of the area to set |
| 5 | 4 | 3S | 4 | UINT32 | offset | The offset into the area |
| 6 | 4 | 4S | 4 | UINT32 | dataSize | The size of the data area |
| 7 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for TPM Owner aurhorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | authNonceOdd | Nonce generated by caller |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | 20 | | | TPM_AUTHDATA | ownerAuth | HMAC key: TPM Owner authorization |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 4 | 2S | | UINT32 | dataSize | The size of the data area |
| 5 | <> | 3S | | BYTE | data | The data to set the area to |
| 6 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TPM_AUTHDATA | ownerAuth | HMAC key: TPM Owner authorization |

**Actions**

1. Set D1 a TPM_NV_DATA_AREA structure to the area pointed to by nvIndex, if not found return TPM_BAD_INDEX
2. If tag = TPM_TAG_RQU_AUTH1_COMMAND then
   a. If D1 -> TPM_NV_PER_OWNERREAD is FALSE return TPM_AUTH_CONFLICT
   b. Validate command and parameters using TPM Owners authorization on error return TPM_AUTHFAIL

3. Else
   a. If D1 -> TPM_PER_NV_AUTHREAD is TRUE return TPM_AUTH_CONFLICT
   b. If D1 -> TPM_PER_NV_OWNERREAD is TRUE return TPM_AUTH_CONFLICT
4. Check that D1 -> pcrInfoRead -> localityAtRelease for TPM_STANY_DATA -> localityModifier is TRUE
   a. For example if TPM_STANY_DATA -> localityModifier was 2 then D1 -> pcrInfo -> localityAtRelease -> TPM_LOC_TWO would have to be TRUE
   b. On error return TPM_BAD_LOCALITY
5. If D1 -> attributes specifies TPM_NV_PER_PPREAD then validate physical presence is asserted if not return TPM_BAD_PRESENCE
6. If D1 -> TPM_NV_PER_READ_STCLEAR then
   a. If D1 -> bReadSTClear is TRUE return TPM_DISABLED_CMD
7. If D1 -> TPM_NV_PER_PCRREAD then
   a. Create P1 a composite hash of the PCR specified by D1 -> pcrSelectionRead
   b. Compate P1 to digestAtRelease return TPM_WRONGPCRVALUE on mismatch
8. If dataSize is 0 then
   a. Set D1 -> bReadSTClear to TRUE
   b. Set data to NULL
9. Else
   a. Set S1 to offset + dataSize
   b. If S1 > D1 -> dataSize return TPM_NOSPACE
   c. Set data to area pointed to by offset
10. Return TPM_SUCCESS

# 20.5 TPM_NV_ReadValueAuth

***Start of informative comment:***

This command requires that the read be authorized by a value set with the blob.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | Tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | Ordinal | Ordinal, TPM_ORD_NV_READVALUEAUTH |
| 4 | 4 | 2S | 4 | TPM_NV_INDEX | nvIndex | The index of the area to set |
| 5 | 4 | 3S | 4 | UNIT32 | offset | The offset from the data area |
| 6 | 4 | 5S | 4 | UINT32 | dataSize | The size of the data area |
| 7 | 4 | | | TPM_AUTHHANDLE | authHandle | authThe auth handle for the NV element authorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | authNonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H1 | 1 | BOOL | authContinueSession | The continue use flag for the authorization handle |
| 10 | 20 | | | TPM_AUTHDATA | authHmac | HMAC key: nv element authorization |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | ordinal, TPM_ORD_NV_READVALUEAUTH |
| 4 | 4 | 3S | 4 | UINT32 | dataSize | The size of the data area |
| 5 | <> | 4S | <> | BYTE | data | The data |
| 6 | 20 | 2H1 | 20 | TPM_NONCE | authNonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | authLastNonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | authContinueSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TPM_AUTHDATA | authHmacOut | HMAC key: nv element authorization |

**Actions**

1. Locate and set D1 to the TPM_NV_DATA_AREA that corresponds to nvIndex, on error return TPM_BAD_INDEX
2. If D1 -> TPM_NV_PER_AUTHREAD is FALSE return TPM_AUTH_CONFLICT
3. Validate authHmac using D1 -> authValue on error return TPM_AUTHFAIL

4. If D1 -> attributes specifies TPM_NV_PER_PPREAD then validate physical presence is asserted if not return TPM_BAD_PRESENCE
5. Check that D1 -> pcrInfoRead -> localityAtRelease for TPM_STANY_DATA -> localityModifier is TRUE
   a. For example if TPM_STANY_DATA -> localityModifier was 2 then D1 -> pcrInfo -> localityAtRelease -> TPM_LOC_TWO would have to be TRUE
   b. On error return TPM_BAD_LOCALITY
6. If D1 -> attributes specifies TPM_NV_PER_PCRREAD then
   a. Create P1 a composite hash of the PCR specified by D1 -> pcrSelectionRead
   b. Compate P1 to digestAtRelease return TPM_WRONGPCRVALUE on mismatch
7. If D1 specifies TPM_NV_PER_READ_STCLEAR then
   a. If D1 -> bReadSTClear is TRUE return TPM_DISABLED_CMD
8. If dataSize is 0 then
   a. Set D1 -> bReadSTClear to TRUE
   b. Set data to NULL
9. Else
   a. Set S1 to offset + dataSize
   b. If S1 > D1 -> dataSize return TPM_NOSPACE
   c. Set data to area pointed to by offset
10. Return TPM_SUCCESS

# 21. Session Management

# 21.1 TPM_KeyControlOwner

***Start of informative comment:***

This command controls some attributes of keys that are stored within the TPM key cache.

OwnerEvict: If this bit is set to true, this key remains in the TPM through all TPM_Startup events. The only way to evict this key is for the TPM Owner to execute this command again, setting the owner control bit to false and then executing TPM_FlushSpecific.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes incl. paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_KeyControlOwner |
| 4 | 4 | | | TPM_KEY_HANDLE | keyHandle | The handle of a loaded key. |
| 5 | 4 | 2S | 4 | UINT32 | bitName | The name of the bit to be modified |
| 6 | 1 | 3S | 1 | BOOL | bitValue | The value to set the bit to |
| 7 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| 8 | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 9 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 10 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 11 | 20 | | 20 | TPM_AUTHDATA | ownerAuth | HMAC authorization: key TPM Owner authorization |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal:TPM_ORD_ KeyControlOwner |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM. |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | HMAC authorization: key TPM Owner authorization |

**Descriptions**

Set an internal bit within the key cache that controls some attribute of a loaded key.

**Actions**

1. Validate the authorization using the owner authorization value, on error return TPM_AUTHFAIL

2. Validate that keyHandle refers to a loaded key, return TPM_INVALID_KEYHANDLE on error.
3. Validate that bitName is valid, return TPM_BAD_MODE on error.
4. If bitName == TPM_KEY_CONTROL_OWNER_EVICT
   a. If bitValue == TRUE
      i. Verify that after this operation at least two key slots will be present within the TPM that can store any type of key both of which do NOT have the OwnerEvict bit set, on error return TPM_NOSPACE
      ii. Verify that for this key handle, ParentPCR is FALSE and Volatile is FALSE, return TPM_BAD_PARAMETER on error.
      iii. Set ownerEvict within the internal key storage structure to TRUE.
   b. Else if bitValue == FALSE
      i. Set ownerEvict within the internal key storage structure to FALSE.
5. Return TPM_SUCCESS

# 21.2 TPM_SaveContext

***Start of informative comment:***

SaveContext saves a loaded resource outside the TPM. After successful execution of the command the TPM automatically releases the internal memory for sessions but leaves keys in place.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SaveAuthContext |
| 4 | 4 | | | TPM_HANDLE | handle | Handle of the resource being saved. |
| 5 | 4 | | | TPM_RESOURCE_TYPE | resourceType | The type of resource that is being saved |
| 6 | 16 | | | BYTE[16] | label | Label for identification purposes |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 4 | | | UINT32 | contextSize | The actual size of the outgoing context blob |
| 5 | <> | | | TPM_CONTEXT_BLOB | contextBlob | The context blob |

**Description**

1. The caller of the function uses the label field to add additional sequencing, anti-replay or other items to the blob. The information does not need to be confidential but needs to be part of the blob integrity.

**Actions**

1. Map V1 to TPM_STANY_DATA
2. Validate that handle points to resource that matches resourceType, return TPM_INVALID_RESOURCE on error
3. Validate that resourceType is a resource from the following list if not return TPM_INVALID_RESOURCE
   a. TPM_RT_KEY
   b. TPM_RT_AUTH
   c. TPM_RT_TRANS

4. Locate the correct nonce
   a. If resourceType is TPM_RT_KEY
      i. If TPM_STCLEAR_DATA -> contextNonceKey is NULLS
         (1) Set TPM_STCLEAR_DATA -> contextNonceKey to the next value from the TPM RNG
      ii. Map N1 to V1 -> contextNonceKey
      iii. If the key has TPM_KEY_CONTROL_OWNER_EVICT set then return TPM_OWNER_CONTROL
   b. Else
      i. If V1 -> contextNonceSession is NULLS
         (1) Set V1 -> contextNonceSession to the next value from the TPM RNG
      ii. Map N1 to V1 -> contextNonceSession
5. Set K1 to V1 -> contextKey
6. Create R1 by putting the sensitive part of the resource pointed to by handle into a structure. The structure is a TPM manufacturer option. The TPM MUST ensure that ALL sensitive information of the resource is included in R1.
7. Create C1 a TPM_CONTEXT_SENSITIVE structure
   a. C1 forms the inner encrypted wrapper for the blob. All saved context blobs MUST include a TPM_CONTEXT_SENSITIVE structure and the TPM_CONTEXT_SENSITIVE_STRUCTURE MUST be encrypted.
   b. Set C1 -> contextNonce to N1
   c. Set C1 -> internalData to R1
8. Create B1 a TPM_CONTEXT_BLOB
   a. Set B1 -> tag to TPM_TAG_CONTEXTBLOB
   b. Set B1 -> resourceType to resourceType
   c. Set B1 -> handle to handle
   d. Set B1 -> integrityDigest to NULL
   e. Set B1 -> label to label
   f. Set B1 -> additionalData to information determined by the TPM manufacturer. This data will help the TPM to reload and reset context. This area MUST NOT hold any data that is sensitive (symmetric IV are fine, prime factors of an RSA key are not).
   g. Set B1 -> additionalSize to the size of additionalData
   h. Set B1 -> sensitiveSize to the size of C1
   i. Set B1 -> sensitiveData to C1
9. If resourceType is TPM_RT_KEY
   a. Set B1 -> contextCount to 0
10. Else
   a. If V1 -> contextCount > $2^{32}$-2 then
      i. Return with TPM_TOOMANYCONTEXTS
   b. Else

i. Increment V1 -> contextCount by 1

ii. Validate that the TPM can still manage the new count value

(1) If the distance between the oldest saved context and the contextCount is too large return TPM_CONTEXT_GAP

iii. Find contextIndex such that V1 -> contextList[contextIndex] equals 0. If not found exit with TPM_NOCONTEXTSPACE

iv. Set V1-> contextList[contextIndex] to V1 -> contextCount

v. Set B1 -> contextCount to V1 -> contextCount

c. The TPM MUST invalidate all information regarding the resource except for information needed for reloading

11. Calculate B1 -> integrityDigest the HMAC of B1 using TPM_PERMANENT_DATA -> tpmProof as the secret
12. Create E1 by encrypting the concatenation of C1 using K1 as the key
    a. Set B1 -> sensitiveSize to the size of E1
    b. Set B1 -> sensitiveData to E1
13. Set contextSize to the size of B1
14. Return B1 in contextBlob
15. If resourceType is not TPM_RT_KEY release interal rescources

# 21.3 TPM_LoadContext

***Start of informative comment:***

LoadContext loads into the TPM a previously saved context. The command returns the type of blob and a handle.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_LoadAuthContext |
| 4 | 1 | | | BOOL | keepHandle | Indication if the handle MUST be preserved |
| 5 | 4 | | | TPM_HANDLE | hintHandle | The hint hand the TPM MAY use to locate a OSAP session tied to a key |
| 6 | 4 | | | UINT32 | contextSize | The size of the following context blob. |
| 7 | <> | | | TPM_CONTEXT_BLOB | contextBlob | The context blob |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 4 | | | TPM_HANDLE | handle | The handle assigned to the resource after it has been successfully loaded. |

**Actions**

1. Map contextBlob to B1, a TPM_CONTEXT_BLOB structure
2. Map V1 to TPM_STANY_DATA
3. Create M1 by decrypting B1 -> sensitiveData using TPM_PERMANENT_DATA -> contextKey
4. Create C1 and R1 by splitting M1 into a TPM_CONTEXT_SENSITIVE structure and internal resource data
5. Check contextNonce
   a. If B1 -> resourceType is NOT TPM_RT_KEY
      i. If C1 -> conextNonce does not equal V1 -> contextNonceSession return TPM_BADCONTEXT
      ii. Validate that the resource pointed to by the context is loaded (i.e. for OSAP the key referenced is loaded) return TPM_RESOURCEMISSING
   b. Else
      i. If C1 -> internalData -> parentPCRStatus is FALSE and C1 -> internalData -> isVolatile is FALSE

(1) Ignore C1 -> contextNonce

ii. else

(1) If C1 -> contextNonce does not equal V1 -> contextNonceKey return TPM_BADCONTEXT

6. Validate the structure
   a. Set H1 to B1 -> integrityDigest
   b. Set B1 -> integrityDigest to NULL
   c. Create H2 the HMAC of B1 using TPM_PERMANENT_DATA -> tpmProof as the HMAC key
   d. If H2 does equal H1 return TPM_BADCONTEXT
7. If keepHandle is TRUE
   a. Set handle to B1 -> handle
   b. If the TPM is unable to restore the handle the TPM MUST return TPM_BADHANDLE
8. Else
   a. The TPM SHOULD attempt to restore the handle but if not possible it MAY set the handle to any valid for R1
9. If B1 -> resourceType is NOT TPM_RT_KEY
   a. Find contextIndex such that V1 -> contextList[contextIndex] equals C1 -> savedCount
   b. If not found then return TPM_BADCONTEXT
   c. Set V1 -> contextList[contextIndex] to 0
10. Process R1 to return the resource back into TPM use

# 22. Eviction

***Start of informative comment:***

The TPM has numerous resources held inside of the TPM that may need eviction. The need for eviction occurs when the number or resources in use by the TPM exceed the available space. For resources that are hard to reload (i.e. keys tied to PCR values) the outside entity should first perform a context save before evicting items.

In version 1.1 there were separate commands to evict separate resource types. This new command set uses the resource types defined for context saving and creates a generic command that will evict all resource types.

***End of informative comment.***

The TPM MUST NOT flush the EK or SRK using this command.

Version 1.2 deprecates the following commands:

- TPM_Terminate_Handle
- TPM_Evict_Key
- TPM_Reset

# 22.1 TPM_FlushSpecific

***Start of informative comment:***

TPM_FlushSpecific flushes from the TPM a specific handle.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_FlushSpecific |
| 4 | 4 | | | TPM_HANDLE | handle | The handle of the item to flush |
| 5 | 4 | | | TPM_RESOURCE_TYPE | resourceType | The type of resource that is being flushed |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation |

**Description**

TPM_FlushSpecific releases the resources associated with the given handle.

**Actions**

1. If resourceType is TPM_RT_CONTEXT
   a. Treat the handle parameter as a contextCount value that MUST be found in contextList, return TPM_BAD_PARAMETER if not found
   b. Set R1 to the contextCount
   c. Validate that R1 points at valid context
2. Else if resourceType is TPM_RT_KEY
   a. Set R1 to the key pointed to by handle
   b. Validate that R1 points at valid key
   c. If R1 -> ownerEvict is TRUE return TPM_KEY_OWNER_CONTROL
3. Else if resourceType is TPM_RT_HASH or TPM_RT_COUNTER
   a. Return TPM_INVALID_RESOURCE
4. Else
   a. Set R1 to the resource pointed to by handle
   b. Validate that resource type and handle point to a valid allocated resource
5. Invalidate R1 and all internal resources allocated to R1
   a. Resources include authorization and transport sessions

# 23. Timing Ticks

***Start of informative comment:***

The TPM timing ticks are always available for use. The association of timing ticks to actual time is a protocol that occurs outside of the TPM. See the design document for details.

The setting of the clock type variable is a one time operation that allows the TPM to be configured to the type of platform that is installed on.

The ability for the TPM to continue to increment the timer ticks across power cycles of the platform is a TPM and platform manufacturer decision.

***End of informative comment.***

# 23.1 TPM_SetTickType

***Start of informative comment:***

This is a one time command that sets the TPM clock type when it is installed on a platform during platform manufacturing.

***End of informative comment.***

**Type**

TPM protected capability. The TPM MAY implement this function in manufacturing and not expose the ordinal to normal operation

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Ordinal: TPM_ORD_SetTickType |
| 4 | 4 | | | TPM_TICKTYPE | tickType | The clock type to be set |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Descriptions**

This command sets the clock type. It is a one-time command that the platform manufacturer uses to indicate to the TPM how the platform will handle the ability to continue to increment the TPM timer ticks between power cycles.

The TPM MAY implement this command in a manner that does not require the use of the command data stream. When implemented in this manner the TPM MUST ensure that the command is only executable once.

**Actions**

1. If TPM_PERMANENT_DATA -> tickType is NULL
   a. Set TPM_PERMANENT_DATA -> tickType

## 23.2 TPM_GetTicks

***Start of informative comment:***

This command returns the current tick count of the TPM.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Ordinal: TPM_ORD_GetTicks |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 36 | | | TPM_CURRENT_TICKS | currentTime | The current time held in the TPM |

**Descriptions**

This command returns the current time held in the TPM. It is the responsibility of the external system to maintain any relation between this time and a UTC value or local real time value.

**Actions**

1. Set T1 to the internal TPM_CURRENT_TICKS structure
2. Return T1 as currentTime.

# 23.3 TPM_TickStampBlob

***Start of informative comment:***

This command applies a time stamp to the passed blob. The TPM makes no representation regarding the blob merely that the blob was present at the TPM at the time indicated.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Ordinal, fixed value of TPM_ORD_TickStampBlob |
| 4 | 4 | | | TPM_KEY_HANDLE | keyHandle | The keyHandle identifier of a loaded key that can perform digital signatures. |
| 5 | 20 | 2s | 20 | TPM_NONCE | antiReplay | Anti replay value to added to signature |
| 6 | 20 | 3s | 20 | TPM_DIGEST | digestToStamp | The digest to perform the tick stamp on |
| 7 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | 20 | | | TPM_AUTHDATA | privAuth | The authorization digest that authorizes the use of keyHandle. HMAC key: key.usageAuth |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Ordinal, fixed value of TPM_ORD_TickStampBlob |
| 4 | 36 | 3S | 36 | TPM_CURRENT_TICKS | currentTicks | The current time according to the TPM |
| 5 | 4 | 4S | 4 | UINT32 | sigSize | The length of the returned digital signature |
| 6 | <> | 5S | <> | BYTE[ ] | sig | The resulting digital signature. |
| 7 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 9 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth |

**Description**

The function performs a digital signature on the hash of digestToStamp and the current tick count.

It is the responsibility of the external system to maintain any relation between tick count and a UTC value or local real time value.

**Actions**

1. The TPM validates the authorization to use the key pointed to by keyHandle.
2. Validate that keyHandle -> keyUsage is TPM_KEY_SIGN, TPM_KEY_IDENTITY or TPM_KEY_LEGACY, if not return the error code TPM_INVALID_KEYUSAGE.
3. Create T1, a TPM_CURRENT_TICKS structure.
4. Create H1 a TPM_SIGN_INFO structure and set the structure defaults
   a. Set H1 -> fixed to “TSTP”
   b. Set H1 -> replay to antiReplay
   c. Create H2 the concatenation of digestToStamp || T1
   d. Set H1 -> dataLen to the length of H2
   e. Set H1 -> data to H2
5. The TPM computes the signature, sig, using the key referenced by keyHandle, using H1 as the information to be signed
6. The TPM returns T1 as currentTime parameter

# 24. Transport Sessions

# 24.1 TPM_EstablishTransport

***Start of informative comment:***

This establishes the transport session. Depending on the attributes specified for the session this may establish shared secrets, encryption keys and session logs. The session will be in use for by the TPM_ExecuteTransport command.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_EstablishTransport |
| 4 | 4 | | | TPM_KEY_HANDLE | encHandle | The handle to the key that encrypted the blob |
| 5 | <> | 2S | <> | TPM_TRANSPORT_PUBLIC | transPublic | The public information describing the transport sesion |
| 6 | 4 | 3S | 4 | UINT32 | secretSize | The size of the secret Area |
| 7 | <> | 4S | <> | BYTE | secret | The encrypted secret area |
| 8 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 9 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 10 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 11 | 20 | | | TPM_AUTHDATA | keyAuth | Authorizatoin. HMAC key: encKey.usageAuth |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_EstablishTransport |
| 4 | 4 | 3S | 4 | TPM_TRANSHANDLE | transHandle | The handle for the transport session |
| 5 | 36 | 4S | 36 | TPM_CURRENT_TICKS | currentTicks | The current tick count |
| 6 | 20 | 5S | 20 | TPM_NONCE | transNonce | The even nonce in use for subsequent execute transport |
| 7 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 9 | 20 | | | TPM_AUTHDATA | resAuth | Authorization. HMAC key: key.usageAuth |

**Description**

This command establishes the transport sessions shared secret. The encryption of the shared secret uses the public key of the key loaded in encKey.

**Actions**

1. If encHandle is TPM_KH_TRANSPORT then
   a. If tag is NOT TPM_TAG_RQU_COMMAND return TPM_BADTAG
   b. If transPublic -> transAttributes specifies TPM_TRANSPORT_ENCRYPT return TPM_BAD_SCHEME
   c. Set A1 to secret. This implies that the secret is a 20 byte authValue.
2. Else
   a. encHandle -> keyType MUST be TPM_KEY_STORAGE or TPM_KEY_LEGACY return TPM_INVALID_KEYUSAGE on error
   b. If tag is NOT TPM_TAG_RQU_AUTH1_COMMAND return TPM_BADTAG
   c. Using keyAuth validate the authorization to use the key and the parameters to the command
   d. Create K1 a TPM_TRANSPORT_AUTH structure by decrypting secret using the key pointed to by encHandle
   e. Validate K1 for tag
   f. Set A1 to K1 -> authData
3. Check if the transPublic->keyParms->algorithmnID is suppored if not return TPM_BAD_KEY_PROPERTY
   a. If transPublic->keyParms->algorithmnID == TPM_ALG_3DES or TPM_ALG_AESXXX check that transPublic->keyParms->encScheme is supported if not return TPM_INAPPROPRIATE_ENC
4. Generate nonces
   a. Generate transLastNonceEven from the TPM RNG
   b. Generate nonceEven from the TPM RNG. This nonce associates with the authorization session in use to establish the session. This is only necessary if tag = TPM_TAG_RQU_AUTH1_CMD
   c. These nonces are separate and do not affect the nonces in use to wrap commands.
5. Create T1 a TPM_TRANSPORT_INTERNAL structure
   a. Ensure that the TPM has sufficient internal space to allocate the transport session, return TPM_NO_SPACE on error
   b. Assign a T1 -> transHandle value. This value is assigned by the TPM and may be any value
   c. Set T1 -> transDigest to NULL
   d. Set T1 -> transAttributes to transPublic -> transAttributes
   e. Set T1-> transEven to transNonce
   f. Set T1 -> authData to A1
6. If T1 -> transAttributes has TPM_TRANSPORT_ENCRYPT
   a. Validate that the TPM can perform the indicated encryption algorithm including mode, on error return TPM_BAD_MODE
   b. Perform any initialization necessary for the indicated algorithm

7. If TPM_STCLEAR_DATA -> currentTicks is not properly initialized
   a. Initialize the TPM_STCLEAR_DATA -> currentTicks
8. Set currentTicks to TPM_STCLEAR_DATA -> currentTicks
9. If T1 -> transAttributes has TPM_TRANSPORT_LOG set then
   a. Create L1 a TPM_TRANSPORT_LOG_IN structure
      i. Set L1 -> parameters to SHA-1 (transPublic)
      ii. Fill other L1 parameters as defined
      iii. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || L1)
   b. Create L2 a TPM_TRANSPORT_LOG_OUT structure
      i. Fill in L2 with information from this command
      ii. Set L2 -> currentTicks to currentTicks, this MUST be the same value that is returned in the currentTicks parameter
      iii. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || L2)
10. If T1 -> transAttributes has TPM_TRANSPORT_EXCLUSIVE set then TPM_STANY_FLAGS -> transportExclusive to TRUE
    a. Any command, other than TPM_ExecuteTransport or TPM_ReleaseTransportSigned, will cause the invalidation of this transport session transHandle
11. Return T1 -> transHandle as transHandle

## 24.2 TPM_ExecuteTransport

***Start of informative comment:***

Delivers a wrapped TPM command to the TPM where the TPM unwraps the command and then executes the command.

ExecuteTransport uses the same rolling nonce paradigm as other authorized TPM commands. The even nonces start in EstablishTransport and change on each invocation of ExecuteTransport.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ExecuteTransport |
| 4 | 4 | 2s | 4 | UINT32 | wrappedCmdSize | Size of the wrapped command |
| 5 | <> | 3s | <> | BYTE[] | wrappedCmd | The wrapped command |
| 6 | 4 | | | TPM_TRANSHANDLE | transHandle | The transport session handle |
| | | 2H1 | 20 | TPM_NONCE | transLastNonceEven | Even nonce previously generated by TPM |
| 7 | 20 | 3H1 | 20 | TPM_NONCE | transNonceOdd | Nonce generated by caller |
| 8 | 1 | 4H1 | 1 | BOOL | continueTranSession | The continue use flag for the authorization handle |
| 9 | 20 | | | TPM_AUTHDATA | transAuth | HMAC for transHandle key: transHandle -> authData |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the ExecuteTransport command. This does not reflect the status of wrapped command. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ExecuteTransport |
| 4 | 8 | 3s | 8 | UINT64 | currentTicks | The current ticks when the command was executed |
| 5 | 4 | 4s | 4 | UINT32 | wrappedCmdSize | Size of the wrapped command |
| 6 | <> | 5s | <> | BYTE[] | wrappedCmd | The wrapped command |
| 7 | 20 | 2H1 | 20 | TPM_NONCE | transNonceEven | Even nonce newly generated by TPM |
| | | 3H1 | 20 | TPM_NONCE | transNonceOdd | Nonce generated by caller |
| 8 | 1 | 4H1 | 1 | BOOL | continueTranSession | The continue use flag for the session |
| 9 | 20 | | | TPM_AUTHDATA | transAuth | HMAC for transHandle key: transHandle -> authData |

**Description**

1. This command executes a TPM command using the transport session.
2. Prior to execution of the wrapped command (action 10 below) failure of the transport session MUST have no effect on the wrapped command. The exception is when the TPM goes into failure mode and return FAILED_SELFTEST for all subsequent commands.
3. After execution of the wrapped command failure of the transport session MUST have an effect on the wrapped command resources. The reason for this is that the transport session will be returning an error code and not reporting any session nonces. The entire wrapped command response is lost so nonces, handles and such are lost to the caller.
4. Execution of the wrapped command (action 10) SHOULD have no effect on the transport session.
    a. The wrapped command SHALL use no resources of the transport session, this includes authorization sessions
    b. If the wrapped command execution returns an error (action 10 below) then the sessions for ExecuteTransport still operate properly.
    c. The exception to this is when the wrapped command causes the TPM to go into failure mode and return TPM_FAILSELFTEST for all subsequent commands
5. Command representation
    a. **********
    b. TAGet | LENet | ORDet | wrappedCmd | AUTHet
    c. **********
    d. And wrappedCmd looks like
    e. **********
    f. TAGw | LENw | ORDw | HANDLESw(o) | DATAw | AUTH1w (o) | AUTH2w (o)
    g. **********

**Actions**

1. Using transHandle locate the TPM_TRANSPORT_INTERNAL structure T1
2. Calculate pointers
    a. Set S1 to DATAw
        i. This pointer is ordinal dependent and requires the execute transport command to parse wrappedCmd
    b. Set L1 to the length of DATAw
        i. If AUTH1w is present no bytes of AUTH1w are in the area pointed to by S1 and L1
3. If T1 -> transAttributes has TPM_TRANSPORT_ENCRYPT set then
    a. If transHandle -> encryptionAlg == MGF1
        i. Using the MGF1 function, create string G1 of length L1. The inputs to the MGF1 are transLastNonceEven, transNonceOdd, “in”, and T1 -> authData. These four values concatenated together form the Z value that is the seed for the MGF1.
        ii. Create C1 by performing an XOR of G1 and wrappedCmd starting at S1.
    b. If the encryption algorithm requires an IV calculate the IV values

i. Using the MGF1 function, create string IV1 with a length set by the block size of the encryption algorithm. The inputs to the MGF1 are transLastNonceEven, transNonceOdd, and “in”. These three values concatenated together form the Z value that is the seed for the MGF1. IV1 is also in use for the output. Note that any terminating characters within the string “in” are ignored, so a total of 42 bytes are hashed.

ii. IV2 is for the output and uses the same algorithm with the input values of transNonceEven, transNonceOdd and “out”.

iii. Blocksize for TPM_ALG_DES is 8

iv. Blocksize for TPM_ALG_AESxxx is 16

v. Create C1 by decrypting the wrappedCmd starting at S1

c. TPM_OSAP and TPM_DSAP have special rules as to which parameters are encrypted see the respective commands for the details

4. Else
   a. Set C1 to wrappedCmd
5. Create H1 the SHA-1 of (ORDw || S1).
   a. S1 MUST point at the decrypted DATAw area of C1
   b. The TPM MAY use this calculation for both execute transport authorization, authorization of the wrapped command and transport log creation
6. Calculate AUTHet
   a. The text field for the HMAC calculation MUST be (ORDet || H1 || transLastNonceEven || transNonceOdd || continueTranSession)
   b. Validate AUTHet using T1 -> authData as the HMAC key, on errors return TPM_AUTHFAIL
   c. Calculate S3 = SHA-1 (RCw || ORDw || DATAOUTw)
      i. The TPM MAY use this calculation for execute transport authorization and transport log out creation
   d. S2 for the command is the SHA-1 (RCet || ORDet || currentticks || S3)
   e. The AUTHet is the HMAC of (S2 || transNonceEven || transNonceOdd || continueTranSession)
7. If TPM_ExecuteTransport requires auditing
   a. Create TPM_AUDIT_EVENT_IN using H1 as the parameters and update auditDigest
   b. On any error return TPM_AUDITFAIL_UNSUCCESSFUL
8. If C1 is from the list of following commands return TPM_NO_WRAP_TRANSPORT
   a. TPM_EstablishTransport
   b. TPM_ExecuteTransport
   c. TPM_ReleaseTransportSigned
   d. TPM_FlushSpecific
   e. TPM_SaveContext
   f. TPM_LoadContext
   g. Any Deprecated command
9. If T1 -> transAttributes has TPM_TRANSPORT_LOG set then

a. Create L2 a TPM_TRANSPORT_LOG_IN structure
b. Set L2 -> parameters to the hash of the input parameters of C1
c. Set L2 -> pubKeyHash to NULL
d. If C1 is a command where the first handle is a key then
    i. Create K2 the hash of the TPM_STORE_PUBKEY structure of the key pointed to by the first key handle
    ii. If C1 is a command where the second handle is a key then
        (1) Create K3 the hash of the TPM_STORE_PUBKEY structure of the key pointed to by the second key handle
        (2) Set K2 to the concatenation K2 || K3
    iii. Set L2 -> pubKeyHash to SHA-1 (K2)

10. Send C1 to the normal TPM command parser, the output is C2 and the return code is RC1
    a. If C1 is a command that is audited then the TPM MUST perform the input and output audit of the command as part of action 10
    b. The TPM MAY use S1 as the data value in the authorization and audit calculations during the execution of C1
11. Set CT1 to currentTicks and return CT1 in the currentTicks output parameter
12. If T1 -> transAttributes has TPM_TRANSPORT_LOG set then
    a. Create L3 a TPM_TRANSPORT_LOG_OUT structure
    b. Set L3 -> parameters to the output parameters of C2
    c. Set L3 -> currentTicks to CT1
    d. Set L3 -> returnCode to RC1
    e. Set T1 -> transDigest to the SHA-1 (T1 -> transDigest || L3)
13. Create N1 the new transNonceEven for the output of the command
14. Calculate S2 the pointer to the DATAw area of C2
    a. Calculate L2 the length of S2 according to the same rules that calculated L1
15. Create H2 the SHA-1 of (ORDw || S2)
16. Calculate AUTHet for output
    a. The text field for the HMAC calculation MUST be (RCet || ORDet || currentticks|| H2 || transNonceEven || transNonceOdd || continueTranSession)
    b. Create AUTHet using T1 -> authData as the HMAC key, on errors return TPM_AUTHFAIL
17. If T1 -> transAttributes has TPM_TRANSPORT_ENCRYPT set then
    a. If transHandle -> encryptionAlg == MGF1
        i. Using the MGF1 function, create string G1 of length L1. The inputs to the MGF1 are transNonceEven, transNonceOdd, “out”, and T1 -> authData. These four values concatenated together form the Z value that is the seed for the MGF1.
        ii. Create E1 by performing an XOR of G1 and C2 starting at S1.
    b. Else

i. Create E1 by encrypting C2 starting at S1 using IV2

18. Else
    a. Set E1 to C2
19. If continueTranSession is FALSE
    a. Invalidate all session data related to transHandle
20. If TPM_ExecuteTranport requires auditing
    a. Create TPM_AUDIT_EVENT_OUT using H2 for the parameters and update the auditDigest
    b. On any errors return TPM_AUDITFAIL_SUCCESSFUL depending on TPM_AUDITFAIL_SUCCESSFUL depending on RC1
21. Return E1 in the wrappedCmd parameter

# 24.3 TPM_ReleaseTransportSigned

***Start of informative comment:***

This command completes the transport session. If logging for this session is turned on, then this command returns a hash of all operations performed during the session along with a digital signature of the hash.

This command serves no purpose if logging is turned off, and results in an error if attempted.

This command uses two authorizations, the key that will sign the log and the authorization from the session. Having the session authorization proves that the requestor that is signing the log is the owner of the session. If this restriction is not put in then an attacker can close the log and sign using their own key.

The hash of the session log includes the information associated with the input phase of execution of the TPM_ReleaseTransportSigned command. It cannot include the output phase information.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ReleaseTransportSigned |
| 4 | 4 | | | TPM_KEY_HANDLE | Key | The key that will perform the signing |
| 5 | 20 | 2s | 20 | TPM_NONCE | antiReplay | Value provided by caller for anti-replay protection |
| 6 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization session to use key |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 7 | 20 | 3H1 | 20 | TPM_NONCE | authNonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 9 | 20 | | | TPM_AUTHDATA | keyAuth | The authorization digest that authorizes the use of key. HMAC key: key -> usageAuth |
| 10 | 4 | | | TPM_TRANSHANDLE | transHandle | The transport session handle |
| | | 2H2 | 20 | TPM_NONCE | tranLastNonceEven | Even nonce in use by execute Transport |
| 11 | 20 | 3H2 | 20 | TPM_NONCE | tranNonceOdd | Nonce supplied by caller for transport session |
| 12 | 1 | 4H2 | 1 | BOOL | tranSession | The continue use flag for the authorization handle |
| 13 | 20 | | | TPM_AUTHDATA | tranAuth | HMAC for transport session key: tranHandle -> authData |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH2_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ReleaseTransportSigned |
| 4 | 36 | 3S | 36 | TPM_CURRENT_TICKS | currentTicks | The current ticks when the command executed |
| 5 | 4 | 4S | 4 | UINT32 | signSize | The size of the signature area |
| 6 | <> | 5S | <> | BYTE[] | signature | The signature of the digest |
| 7 | 20 | 2H1 | 20 | TPM_NONCE | authNonceEven | Even nonce newly generated by TPM |
| | | 3H1 | 20 | TPM_NONCE | authNonceOdd | Nonce generated by caller |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the session |
| 9 | 20 | | | TPM_AUTHDATA | keyAuth | HMAC: key -> authData |
| 10 | 20 | 2H2 | 20 | TPM_NONCE | transNonceEven | Even nonce newly generated by TPM |
| | | 3H2 | 20 | TPM_NONCE | transNonceOdd | Nonce generated by caller |
| 11 | 1 | 4H2 | 1 | BOOL | continueTranSession | The continue use flag for the session |
| 12 | 20 | | | TPM_AUTHDATA | transAuth | HMAC: tranHandle -> authData |

**Description**

This command releases a transport session and signs the transport log

**Actions**

1. Using transHandle locate the TPM_TRANSPORT_INTERNAL structure T1
2. Using key -> authData validate the command and parameters, on error return TPM_AUTHFAIL
3. Using tranHandle -> authData validate the command and parameters, on error return TPM_AUTH2FAIL
4. Generate a new authNonceEven and transAuthEven from the TPM internal RNG
5. If T1 -> transAttributes has TPM_TRANSPORT_LOG set then
   a. Create A1 a TPM_TRANSPORT_LOG_OUT structure
   b. Set A1 -> ordinal to TPM_ORD_ReleaseTransportSigned
   c. Set A1 -> parameters to the SHA-1 (ordinal || antiReplay)
   d. Set A1 -> currentTicks to TPM_STCLEAR_DATA -> currentTicks
   e. Set A1 -> returnCode to 0
   f. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || A1)
6. Else
   a. Return TPM_BAD_MODE
7. Create H1 a TPM_SIGN_INFO structure and set the structure defaults

a. Set H1 -> fixed to “TRAN”
b. Set H1 -> replay to antiReplay
c. Set H1 -> data to T1 -> transDigest
d. Sign H1 using the key pointed to by key

8. Invalidate all session data related to T1
9. Create response HMAC
10. Return TPM_SUCCESSFUL

# 25. Monotonic Counter

## 25.1 TPM_CreateCounter

***Start of informative comment:***

This command creates the counter but does not select the counter. Counter creation assigns an authorization value to the counter and sets the counters original start value. The original start value is the current internal base value plus one. Setting the new counter to the internal base avoids attacks on the system that are attempting to use old counter values.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes incl. paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CreateCounter |
| 4 | 20 | 2S | 20 | TPM_ENCAUTH | authData | The encrypted auth data for the new counter |
| 5 | 4 | 3s | 4 | BYTE | label | Label to associate with counter |
| 7 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | 20 | | 20 | TPM_AUTHDATA | ownerAuth | Authorization TPM Owner authorization. |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CreateCounter |
| 4 | 4 | 3s | 4 | TPM_COUNT_ID | countID | The handle for the counter |
| 5 | 10 | 4S | 10 | TPM_COUNTER_VALUE | counterValue | The starting counter value |
| 6 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, |
| 8 | 20 | | 20 | TPM_AUTHDATA | resAuth | Authorization. HMAC key: TPM Owner authorization. |

**Description**

This command creates a new monotonic counter. The TPM MUST support a minimum of 4 concurrent counters.

**Actions**

The TPM SHALL do the following:

1. Using the authHandle field, validate the owner's authorization to execute the command and all of the incoming parameters. The authorization session MUST be OSAP.
2. Ignore continueAuthSession on input and set continueAuthSession to FALSE on output
3. Create X1 the SHA-1 result of the concatenation of (authHandle -> sharedSecret || authLastNonceEven)
   a. Create A1 the XOR of encAuth and X1
4. Validate that there is sufficient internal space in the TPM to create a new counter. If there is insufficient space the command returns an error.
5. Increment the internal base counter.
6. Set the counter to the max counter value.
7. Create a countID

# 25.2 TPM_IncrementCounter

***Start of informative comment:***

This authorized command increments the indicated counter by one. Once a counter has been incremented then all subsequent increments must be for the same handle until a successful TPM_Startup(ST_CLEAR) is executed.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_IncrementCounter |
| 4 | 4 | 2s | 4 | TPM_COUNT_ID | countID | The handle of a valid counter |
| 5 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for counter authorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 6 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 8 | 20 | | | TPM_AUTHDATA | counterAuth | The authorization digest that authorizes the use of countID. HMAC key: countID -> authData |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_IncrementCounter |
| 5 | 10 | 3S | 10 | TPM_COUNTER_VALUE | count | The counter value |
| 6 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: countID -> authData |

**Description**

This function increments the counter by 1.

The TPM MAY implement increment throttling to avoid burn problems

**Actions**

1. If TPM_STCLEAR_DATA -> countID is NULL

   a. Set TPM_STCLEAR_DATA -> countID to countID
2. else
   a. If TPM_STCLEAR_DATA -> countID does not equal countID
      i. Return TPM_BAD_COUNTER
3. Increments the counter by 1
4. Return new count value in count

# 25.3 TPM_ReadCounter

***Start of informative comment:***

Reading the counter provides the caller with the current number in the sequence.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes incl. paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ReadCounter |
| 4 | 4 | | | TPM_COUNT_ID | countID | ID value of the counter |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 10 | | | TPM_COUNTER_VALUE | count | The counter value |

**Description**

This returns the current value for the counter indicated. The counter MAY be any valid counter.

**Actions**

1. Validate that countID points to a valid counter. Return TPM_BAD_COUNTER on error.
2. Return count

# 25.4 TPM_ReleaseCounter

***Start of informative comment:***

This command releases a counter such that no reads or increments of the indicated counter will succeed.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ReleaseCounter |
| 4 | 4 | 2s | 4 | TPM_COUNT_ID | countID | ID value of the counter |
| 5 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for countID authorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 6 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce associated with countID |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 8 | 20 | | | TPM_AUTHDATA | counterAuth | The authorization digest that authorizes the use of countID. HMAC key: countID -> authData |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ReleaseCounter |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: countID -> authData |

**Actions**

The TPM uses countID to locate a valid counter.

1. Authenticate the command and the paramenters using the authorization data pointed to by countID. Return TPM_AUTHFAIL on error
2. The TPM invalidates all internal information regarding the counter. This includes releasing countID such that any subsequent attempts to use countID will fail.

# 25.5 TPM_ReleaseCounterOwner

***Start of informative comment:***

This command releases a counter such that no reads or increments of the indicated counter will succeed.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ReleaseCounterOwner |
| 4 | 4 | 2s | 4 | TPM_COUNT_ID | countID | ID value of the counter |
| 5 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 6 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 8 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest that authorizes the inputs. HMAC key: TPM _owner->authData |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ReleaseCounterOwner |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: TPM Owner -> authData |

**Description**

This invalidates all information regarding a counter.

**Actions**

1. Validate that ownerAuth properly authorizes the command and parameters
2. The TPM uses countID to locate a valid counter. Return TPM_BAD_COUNTER if not found.
3. The TPM invalidates all internal information regarding the counter. This includes releasing countID such that any subsequent attempts to use countID will fail.

# 26. DAA commands

## 26.1 TPM_DAA_Join

***Start of informative comment:***

TPM_DAA_Join is the process that establishes the DAA parameters in the TPM for a specific DAA issuing authority.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes incl. paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_DAA_Join. |
| 4 | 4 | | | TPM_HANDLE | handle | Session handle |
| 5 | 1 | 2S | 1 | BYTE | stage | Processing stage of join |
| 6 | 4 | 3S | 4 | UINT32 | inputSize0 | Size of inputData0 for this stage of JOIN |
| 7 | <> | 4S | <> | BYTE[] | inputData0 | Data to be used by this capability |
| 8 | 4 | 5S | 4 | UINT32 | inputSize1 | Size of inputData1 for this stage of JOIN |
| 9 | <> | 6S | <> | BYTE[] | inputData1 | Data to be used by this capability |
| 10 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization |
| | | 2 H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 11 | 20 | 3 H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 12 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 13 | 20 | | 20 | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes incl. paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation |
| 4 | 4 | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_DAA_Join. |
| 5 | 4 | 3S | 4 | UINT32 | outputSize | Size of outputData |
| 6 | <> | 4S | <> | BYTE[] | outputData | Data produced by this capability |
| 7 | 20 | 2 H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 9 | 20 | | 20 | TPM_AUTHDATA | resAuth | Authorization HMAC key: ownerAuth. |

**Description**

This table summaries the input, output and saved data that is associated with each stage of processing.

| Stage | Input Data0 | Input Data1 | Operation | Output Data | Scratchpad |
|---|---|---|---|---|---|
| 0 | DAA_count (used as # repetitions of stage 1) | NULL | initialise | Session Handle | NULL |
| 1 | n0 | signatureValue | rekeying | NULL | n0 |
| 2 | DAA_issuerSettings | signatureValue | issuer settings | NULL | NULL |
| 3 | DAA_count | DAA_generic_n1 | DAA_join_uo, DAA_join_u1 | NULL | NULL |
| 4 | DAA_generic_R0 | DAA_generic_n | P1=R0^f0 modn | NULL | P1 |
| 5 | DAA_generic_R1 | DAA_generic_n | P2 = P1*(R1^f1) modn | NULL | P2 |
| 6 | DAA_generic_S0 | DAA_generic_n | P3 = P2*(S0^u0) modn | NULL | P3 |
| 7 | DAA_generic_S1 | DAA_generic_n | U = P3*(S1^u1) modn | U | NULL |
| 8 | NE | NULL | U2 | U2 | NULL |
| 9 | DAA_generic_R0 | DAA_generic_n | P1=R0^r0 modn | NULL | P1 |
| 10 | DAA_generic_R1 | DAA_generic_n | P2 = P1*(R1^r1) modn | NULL | P2 |
| 11 | DAA_generic_S0 | DAA_generic_n | P3 = P2*(S0^r2) modn | NULL | P3 |
| 12 | DAA_generic_S1 | DAA_generic_n | P4 = P3*(S1^r3) modn | P4 | NULL |
| 13 | DAA_generic_gamma | w | w1 = w^q mod gamma | NULL | w |
| 14 | DAA_generic_gamma | NULL | E = w^f mod gamma | E | w |
| 15 | DAA_generic_gamma | NULL | r = r0 + (2^power0)*r1 modq, E1 = w^r mod gamma | E1 | NULL |
| 16 | c1 | NULL | c = hash(c1 \|\| NT) | c | NULL |
| 17 | NULL | NULL | s0 = r0 + c*f0 | s0 | NULL |
| 18 | NULL | NULL | s1 = r1 + c*f1 | s1 | NULL |
| 19 | NULL | NULL | s2 = r2 + c*u0 mod 2^power1 | s2 | NULL |
| 20 | NULL | NULL | s12 = r2 + c*u0 >> power1 | NULL | s12 |
| 21 | NULL | NULL | s3 = r3 + c*u1 + s12 | s3 | NULL |
| 22 | u2 | NULL | v0 = u2 + u0 mod 2^power1 v10 = u2 + u0 >> power1 | enc(v0) | v10 |
| 23 | u3 | NULL | V1 = u3 + u1 + v10 | enc(v1) | NULL |
| 24 | NULL | NULL | enc(DAA_tpmSpecific) | enc(DAA_tpmSpecific) | NULL |

**Actions**

A Trusted Platform Module that receives a valid TPM_DAA_Join command SHALL:

1. Use ownerAuth to verify that the Owner authorized all TPM_DAA_Join input parameters.
2. Any error return results in the TPM invalidating all resources associated with the join

**Stages**

0. If stage==0
   a. Determine that sufficient resources are available to perform a DAA_Join.
      i. The TPM MUST support sufficient resources to perform one (1) DAA_Join. The TPM MAY support addition DAA_Join sessions
      ii. Insufficient resources return TPM_DAA_RESOURCES
   b. Set all fields in DAA_issuerSettings = NULL
   c. set all fields in DAA_tpmSpecific = NULL
   d. set all fields in DAA_session = NULL
   e. Verify that sizeOf(inputData0) == sizeOf(DAA_tpmSpecific -> DAA_count) and return error TPM_DAA_INPUT_DATA0 on mismatch
   f. Verify that inputData0 > 0, and return error TPM_DAA_INPUT_DATA0 on mismatch
   g. Set DAA_tpmSpecific -> DAA_count = inputData0
   h. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific)
   i. set DAA_session -> DAA_stage = 1
   j. Assign session handle for TPM_Join
   k. set outputData = new session handle
   l. return TPM_SUCCESS
1. If stage==1
   a. Verify that DAA_session ->DAA_stage==1. Return TPM_DAA_STAGE and flush handle on mismatch
   b. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch
   c. Verify that sizeOf(inputData0) == TPM_PUBKEY and return error TPM_DAA_INPUT_DATA0 on mismatch
   d. If DAA_session -> DAA_scratch == NULL:
      i. Set DAA_session -> DAA_scratch = inputData0
      ii. set DAA_joinSession -> DAA_digest_n0 = SHA1(DAA_session -> DAA_scratch)
      iii. set DAA_tpmSpecific -> DAA_rekey = SHA1(TPM_DAA_TPM_SEED || DAA_joinSession -> DAA_digest_n0)
   e. Else (If DAA_session -> DAA_scratch != NULL):
      i. Set signedData = inputData0
      ii. Verify that sizeOf(inputData1) == DAA_SIZE_issuerModulus and return error TPM_DAA_INPUT_DATA1 on mismatch
      iii. Set signatureValue = inputData1

iv. Use the RSA key == [DAA_session -> DAA_scratch] to verify that signatureValue is a signature on signedData, and return error TPM_DAA_ISSUER_VALIDITY on mismatch

v. Set DAA_session -> DAA_scratch = signedData

vi. Set DAA_joinSession -> DAA_digest_n0 = SHA1(DAA_session -> DAA_scratch)

f. Decrement DAA_tpmSpecific -> DAA_count by 1 (unity)

g. If DAA_tpmSpecific -> DAA_count ==0:

i. increment DAA_Session -> DAA_Stage by 1

h. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific || DAA_joinSession)

i. set outputData = NULL

j. return TPM_SUCCESS

2. If stage==2

a. Verify that DAA_session ->DAA_stage==2. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

c. Verify that sizeOf(inputData0) == sizeOf(TPM_DAA_ISSUER) and return error TPM_DAA_INPUT_DATA0 on mismatch

d. Set DAA_issuerSettings = inputData0. Verify that all fields in DAA_issuerSettings are present and return error TPM_DAA_INPUT_DATA0 if not.

e. Verify that sizeOf(inputData1) == DAA_SIZE_issuerModulus and return error TPM_DAA_INPUT_DATA1 on mismatch

f. Set signatureValue = inputData1

g. Set signedData = SHA1(DAA_joinSession -> DAA_digest_n0 ||DAA_issuerSettings)

h. Use the RSA key [DAA_session -> DAA_scratch] to verify that signatureValue is a signature on signedData, and return error TPM_DAA_ISSUER_VALIDITY on mismatch

i. Set DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)

j. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific || DAA_joinSession)

k. Set DAA_session -> DAA_scratch = NULL

l. increment DAA_session -> DAA_stage by 1

m. return TPM_SUCCESS

3. If stage==3

a. Verify that DAA_session ->DAA_stage==3. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Verify that sizeOf(inputData0) == sizeOf(DAA_tpmSpecific -> DAA_count) and return error TPM_DAA_INPUT_DATA0 on mismatch

e. Set DAA_tpmSpecific -> DAA_count = inputData0

f. Set DAA_generic_n1 = inputData1

g. Verify that SHA-1(DAA_generic_n1) == DAA_issuerSettings -> DAA_digest_n1 and return error TPM_DAA_INPUT_DATA1 on mismatch
h. obtain random data from the RNG and store it as DAA_joinSession -> DAA_join_u0
i. obtain DAA_SIZE_u11 bytes of random data from the RNG and label them as u11
j. set DAA_joinSession -> DAA_join_u1 = u11 mod (DAA_generic_n1)
k. set outputData = NULL
l. increment DAA_session -> DAA_stage by 1
m. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific || DAA_joinSession)
n. return TPM_SUCCESS

4. If stage==4,
   a. Verify that DAA_session ->DAA_stage==4. Return TPM_DAA_STAGE and flush handle on mismatch
   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
   c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
   d. Set DAA_generic_R0 = inputData0
   e. Verify that SHA-1(DAA_generic_R0) == DAA_issuerSettings -> DAA_digest_R0 and return error TPM_DAA_INPUT_DATA0 on mismatch
   f. Set DAA_generic_n = inputData1
   g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1on mismatch
   h. Set X = DAA_generic_R0
   i. Set n = DAA_generic_n
   j. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0 ) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1 ) mod DAA_issuerSettings -> DAA_generic_q.
   k. Set f0 = f mod 2^DAA_power0 (erase all but the lowest DAA_power0 bits of f)
   l. Set DAA_session -> DAA_scratch = (X^f0) mod n
   m. set outputData = NULL
   n. increment DAA_session -> DAA_stage by 1
   o. return TPM_SUCCESS

5. If stage==5
   a. Verify that DAA_session ->DAA_stage==5. Return TPM_DAA_STAGE and flush handle on mismatch
   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
   c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
   d. Set DAA_generic_R1 = inputData0

e. Verify that SHA-1(DAA_generic_R1) == DAA_issuerSettings -> DAA_digest_R1 and return error TPM_DAA_INPUT_DATA0on mismatch

f. Set DAA_generic_n = inputData1

g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1on mismatch

h. Set X = DAA_generic_R1

i. Set n = DAA_generic_n

j. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0 ) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1 ) mod DAA_issuerSettings -> DAA_generic_q.

k. Shift f right by DAA_power0 bits (discard the lowest DAA_power0 bits) and label the result f1

l. Set Z = DAA_session -> DAA_scratch

m. Set DAA_session -> DAA_scratch = Z*(X^f1) mod n

n. set outputData = NULL

o. increment DAA_session -> DAA_stage by 1

p. return TPM_SUCCESS

6. If stage==6

a. Verify that DAA_session ->DAA_stage==6. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_generic_S0 = inputData0

e. Verify that SHA-1(DAA_generic_S0) == DAA_issuerSettings -> DAA_digest_S0 and return error TPM_DAA_INPUT_DATA0 on mismatch

f. Set DAA_generic_n = inputData1

g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1on mismatch

h. Set X = DAA_generic_S0

i. Set n = DAA_generic_n

j. Set Z = DAA_session -> DAA_scratch

k. Set Y = DAA_joinSession -> DAA_join_u0

l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n

m. set outputData = NULL

n. increment DAA_session -> DAA_stage by 1

o. return TPM_SUCCESS

7. If stage==7

a. Verify that DAA_session ->DAA_stage==7. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
d. Set DAA_generic_S1 = inputData0
e. Verify that SHA-1(DAA_generic_S1) == DAA_issuerSettings -> DAA_digest_S1 and return error TPM_DAA_INPUT_DATA0 on mismatch
f. Set DAA_generic_n = inputData1
g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1on mismatch
h. Set X = DAA_generic_S1
i. Set n = DAA_generic_n
j. Set Y = DAA_joinSession -> DAA_join_u1
k. Set Z = DAA_session -> DAA_scratch
l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n
m. Set DAA_session -> DAA_digest to the SHA-1 (DAA_session -> DAA_scratch || DAA_tpmSpecific -> DAA_count || DAA_joinSession -> DAA_digest_n0)
n. set outputData = DAA_session -> DAA_scratch
o. set DAA_session -> DAA_scratch = NULL
p. increment DAA_session -> DAA_stage by 1
q. return TPM_SUCCESS

8. If stage==8
   a. Verify that DAA_session ->DAA_stage==8. Return TPM_DAA_STAGE and flush handle on mismatch
   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
   c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
   d. Verify inputSize0 == DAA_SIZE_NE and return error TPM_DAA_INPUT_DATA0 on mismatch
   e. Set NE = decrypt(inputData0, privEK)
   f. set outputData = SHA-1(DAA_session -> DAA_digest || NE)
   g. set DAA_session -> DAA_digest = NULL
   h. increment DAA_session -> DAA_stage by 1
   i. return TPM_SUCCESS
9. If stage==9
   a. Verify that DAA_session ->DAA_stage==9. Return TPM_DAA_STAGE and flush handle on mismatch
   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
   c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_generic_R0 = inputData0
e. Verify that SHA-1(DAA_generic_R0) == DAA_issuerSettings -> DAA_digest_R0 and return error TPM_DAA_INPUT_DATA0 on mismatch
f. Set DAA_generic_n = inputData1
g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1on mismatch
h. obtain random data from the RNG and store it as DAA_session -> DAA_contextSeed
i. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and label them Y
j. Set X = DAA_generic_R0
k. Set n = DAA_generic_n
l. Set DAA_session -> DAA_scratch = (X^Y) mod n
m. set outputData = NULL
n. increment DAA_session -> DAA_stage by 1
o. return TPM_SUCCESS

10. If stage==10
    a. Verify that DAA_session ->DAA_stage==10. Return TPM_DAA_STAGE and flush handle on mismatch h
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. Set DAA_generic_R1 = inputData0
    e. Verify that SHA-1(DAA_generic_R1) == DAA_issuerSettings -> DAA_digest_R1 and return error TPM_DAA_INPUT_DATA0 on mismatch
    f. Set DAA_generic_n = inputData1
    g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1on mismatch
    h. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and label them Y
    i. Set X = DAA_generic_R1
    j. Set n = DAA_generic_n
    k. Set Z = DAA_session -> DAA_scratch
    l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n
    m. set outputData = NULL
    n. increment DAA_session -> DAA_stage by 1
    o. return TPM_SUCCESS
11. If stage==11
    a. Verify that DAA_session ->DAA_stage==11. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
d. Set DAA_generic_S0 = inputData0
e. Verify that SHA-1(DAA_generic_S0) == DAA_issuerSettings -> DAA_digest_S0 and return error TPM_DAA_INPUT_DATA0 on mismatch
f. Set DAA_generic_n = inputData1
g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1 on mismatch
h. obtain DAA_SIZE_r2 bits from MGF1("r2", DAA_session -> DAA_contextSeed), and label them Y
i. Set X = DAA_generic_S0
j. Set n = DAA_generic_n
k. Set Z = DAA_session -> DAA_scratch
l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n
m. set outputData = NULL
n. increment DAA_session -> DAA_stage by 1
o. return TPM_SUCCESS

12. If stage==12
    a. Verify that DAA_session ->DAA_stage==12. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings ) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. Set DAA_generic_S1 = inputData0
    e. Verify that SHA-1(DAA_generic_S1) == DAA_issuerSettings -> DAA_digest_S1 and return error TPM_DAA_INPUT_DATA0 on mismatch
    f. Set DAA_generic_n = inputData1
    g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1 on mismatch
    h. obtain DAA_SIZE_r3 bits from MGF1("r3", DAA_session -> DAA_contextSeed), and label them Y
    i. Set X = DAA_generic_S1
    j. Set n = DAA_generic_n
    k. Set Z = DAA_session -> DAA_scratch
    l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n
    m. set outputData = DAA_session -> DAA_scratch
    n. Set DAA_session -> DAA_scratch = NULL
    o. increment DAA_session -> DAA_stage by 1
    p. return TPM_SUCCESS
13. If stage==13

a. Verify that DAA_session->DAA_stage==13. Return TPM_DAA_STAGE and flush handle on mismatch
b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
d. Set DAA_generic_gamma = inputData0
e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings -> DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch
f. Verify that inputSize1 == DAA_SIZE_w and return error TPM_DAA_INPUT_DATA1 on mismatch
g. Set w = inputData1
h. Set w1 = w^( DAA_issuerSettings -> DAA_generic_q) mod (DAA_generic_gamma)
i. If w1 != 1 (unity), return error TPM_DAA_WRONG_W
j. Set DAA_session -> DAA_scratch = w
k. set outputData = NULL
l. increment DAA_session -> DAA_stage by 1
m. return TPM_SUCCESS.

14. If stage==14
    a. Verify that DAA_session ->DAA_stage==14. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings ) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. Set DAA_generic_gamma = inputData0
    e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings -> DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch
    f. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0 ) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1 ) mod DAA_issuerSettings -> DAA_generic_q.
    g. Set E = ((DAA_session -> DAA_scratch)^f) mod (DAA_generic_gamma).
    h. Set outputData = E
    i. increment DAA_session -> DAA_stage by 1
    j. return TPM_SUCCESS.
15. If stage==15
    a. Verify that DAA_session ->DAA_stage==15. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. Set DAA_generic_gamma = inputData0

e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings -> DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch
f. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and label them r0
g. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and label them r1
h. set r = r0 + 2^DAA_power0 * r1 mod (DAA_issuerSettings -> DAA_generic_q).
i. set E1 = ((DAA_session -> DAA_scratch)^r) mod (DAA_generic_gamma).
j. Set DAA_session -> DAA_scratch = NULL
k. Set outputData = E1
l. increment DAA_session -> DAA_stage by 1
m. return TPM_SUCCESS.

16. If stage==16
    a. Verify that DAA_session ->DAA_stage==16. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. Verify that inputSize0 == sizeOf(TPM_DIGEST) and return error TPM_DAA_INPUT_DATA0 on mismatch
    e. Set DAA_session -> DAA_digest = inputData0
    f. obtain DAA_SIZE_NT bits from the RNG and label them NT
    g. Set DAA_session -> DAA_digest to the SHA-1 ( DAA_session -> DAA_digest || NT )
    h. Set outputData = NT
    i. increment DAA_session -> DAA_stage by 1
    j. return TPM_SUCCESS.
17. If stage==17
    a. Verify that DAA_session ->DAA_stage==17. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and label them r0
    e. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0 ) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1 ) mod DAA_issuerSettings -> DAA_generic_q.
    f. Set f0 = f mod 2^DAA_power0 (erase all but the lowest DAA_power0 bits of f)
    g. Set s0 = r0 + (DAA_session -> DAA_digest) * f0 in Z
    h. set outputData = s0
    i. increment DAA_session -> DAA_stage by 1
    j. return TPM_SUCCESS

18. If stage==18
    a. Verify that DAA_session ->DAA_stage==18. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and label them r1
    e. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0 ) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1 ) mod DAA_issuerSettings -> DAA_generic_q.
    f. Shift f right by DAA_power0 bits (discard the lowest DAA_power0 bits) and label the result f1
    g. Set s1 = r1 + (DAA_session -> DAA_digest)* f1 in Z
    h. set outputData = s1
    i. increment DAA_session -> DAA_stage by 1
    j. return TPM_SUCCESS
19. If stage==19
    a. Verify that DAA_session ->DAA_stage==19. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. obtain DAA_SIZE_r2 bits from MGF1("r2", DAA_session -> DAA_contextSeed), and label them r2
    e. Set s2 = r2 + (DAA_session -> DAA_digest)*( DAA_joinSession -> DAA_join_u0) mod 2^DAA_power1 (Erase all but the lowest DAA_power1 bits of s2)
    f. Set DAA_session -> DAA_scratch = s2
    g. set outputData = s2
    h. increment DAA_session -> DAA_stage by 1
    i. return TPM_SUCCESS
20. If stage==20
    a. Verify that DAA_session ->DAA_stage==20. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. obtain DAA_SIZE_r2 bits from MGF1("r2", DAA_session -> DAA_contextSeed), and label them r2
    e. Set s12 = r2 + (DAA_session -> DAA_digest)*( DAA_joinSession -> DAA_join_u0)
    f. Shift s12 right by DAA_power1 bit (discard the lowest DAA_power1 bits).
    g. Set DAA_session -> DAA_scratch = s12
    h. Set outputData = DAA_session -> DAA_digest

i. increment DAA_session -> DAA_stage by 1

j. return TPM_SUCCESS

21. If stage==21

a. Verify that DAA_session ->DAA_stage==21. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. obtain DAA_SIZE_r3 bits from MGF1("r3", DAA_session -> DAA_contextSeed), and label them r3

e. Set s3 = r3 + (DAA_session -> DAA_digest)*( DAA_joinSession -> DAA_join_u1) + (DAA_session -> DAA_scratch).

f. Set DAA_session -> DAA_scratch = NULL

g. set outputData = s3

h. increment DAA_session -> DAA_stage by 1

i. return TPM_SUCCESS

22. If stage==22

a. Verify that DAA_session ->DAA_stage==22. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Verify inputSize0 == DAA_SIZE_u2 and return error TPM_DAA_INPUT_DATA0 on mismatch

e. Set u2 = inputData0

f. Set v0 = u2 + (DAA_joinSession -> DAA_join_u0) mod 2^DAA_power1 (Erase all but the lowest DAA_power1 bits of v0).

g. Set DAA_tpmSpecific -> DAA_digest_v0 = SHA-1(v0)

h. Set v10 = u2 + (DAA_joinSession -> DAA_join_u0) in Z.

i. Shift v10 right by DAA_power1 bits (erase the lowest DAA_power1 bits).

j. Set DAA_session ->DAA_scratch = v10

k. Set outputData

i. Fill in TPM_DAA_BLOB with a type of TPM_RT_DAA_V0 and encrypt the v0 parameters

ii. set outputData to the encrypted TPM_DAA_BLOB

l. increment DAA_session -> DAA_stage by 1

m. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific || DAA_joinSession)

n. return TPM_SUCCESS

23. If stage==23

a. Verify that DAA_session ->DAA_stage==23. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
d. Verify inputSize0 == DAA_SIZE_u3 and return error TPM_DAA_INPUT_DATA0 on mismatch
e. Set u3 = inputData0
f. Set v1 = u3 + DAA_joinSession -> DAA_join_u1 + DAA_session ->DAA_scratch
g. Set DAA_tpmSpecific -> DAA_digest_v1 = SHA-1(v1)
h. Set outputData
i. Fill in TPM_DAA_BLOB with a type of TPM_RT_DAA_V1 and encrypt the v1 parameters
j. set outputData to the encrypted TPM_DAA_BLOB
    i. Set DAA_session ->DAA_scratch = NULL
    ii. increment DAA_session -> DAA_stage by 1
k. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific || DAA_joinSession)
l. return TPM_SUCCESS

24. If stage==24
    a. Verify that DAA_session ->DAA_stage==24. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific || DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. set outputData = enc(DAA_tpmSpecific)
    e. return TPM_SUCCESS
25. If stage > 24, return error: TPM_DAA_STAGE

## 26.2 TPM_DAA_Sign

TPM protected capability; user must provide authorizations from the TPM Owner.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes incl. paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_DAA_SIGN. |
| 4 | 4 | | | TPM_HANDLE | handle | Handle to the sign session |
| 5 | 1 | 2S | 1 | BYTE | stage | Stage of the sign process |
| 6 | 4 | 3S | 4 | UINT32 | inputSize0 | Size of inputData0 for this stage of DAA_Sign |
| 7 | <> | 4S | <> | BYTE[] | inputData0 | Data to be used by this capability |
| 8 | 4 | 5S | 4 | UINT32 | inputSize1 | Size of inputData1 for this stage of DAA_Sign |
| 9 | <> | 6S | <> | BYTE[] | inputData1 | Data to be used by this capability |
| 10 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization |
| | | 2 H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 11 | 20 | 3 H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 12 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 13 | 20 | | 20 | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes incl. paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation |
| 4 | 4 | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal:TPM_ORD_DAA_SIGN. |
| 5 | 4 | 3S | 4 | UINT32 | outputSize | Size of outputData |
| 6 | <> | 4S | <> | BYTE[] | outputData | Data produced by this capability |
| 7 | 20 | 2 H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 9 | 20 | | 20 | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Description**

This table summaries the input, output and saved data that is associated with each stage of processing.

**Stage** **Input Data0** **Input Data1** **Operation** **Output Data** **Scratchpad**

| | | | | | |
|---|---|---|---|---|---|
| 0 | DAA_issuerSettings | NULL | initialise | handle | NULL |
| 1 | enc(DAA_tpmSpecific) | NULL | initialise | NULL | NULL |
| 2 | DAA_generic_R0 | DAA_generic_n | P1=R0^r0 modn | NULL | P1 |
| 3 | DAA_generic_R1 | DAA_generic_n | P2 = P1*(R1^r1) modn | NULL | P2 |
| 4 | DAA_generic_S0 | DAA_generic_n | P3 = P2*(S0^r2) modn | NULL | P3 |
| 5 | DAA_generic_S1 | DAA_generic_n | T = P3*(S1^r4) modn | T | NULL |
| 6 | DAA_generic_gamma | w | w1 = w^q mod gamma | NULL | w |
| 7 | DAA_generic_gamma | NULL | E = w^f mod gamma | E | w |
| 8 | DAA_generic_gamma | NULL | r = r0 + (2^power0)*r1 modq,<br>E1 = w^r mod gamma | E1 | NULL |
| 9 | c1 | NULL | c = hash(c1 \|\| NT) | NT | NULL |
| 10 | b (selector) | m or handle to AIK | c = hash(c \|\| 1 \|\| m)<br>or<br>c = hash(c \|\| 0 \|\| AIK-modulus) | c | NULL |
| 11 | NULL | NULL | s0 = r0 + c*f0 | s0 | NULL |
| 12 | NULL | NULL | s1 = r1 + c*f1 | s1 | NULL |
| 13 | enc(v0) | NULL | s2 = r2 + c*v0<br>mod 2^power1 | s2 | NULL |
| 14 | enc(v0) | NULL | s12 = r2 + c*v0<br>>> power1 | NULL | s12 |
| 15 | enc(v1) | NULL | s3 = r4 + c*v1 + s12 | s3 | NULL |

When a TPM receives an Owner authorized command to input enc(DAA_tpmSpecific) or enc(v0) or enc(v1), the TPM MUST determine that such data was created by the same TPM and the same Owner. Loading one of these wrapped blobs does not require authorization, since correct blobs were created by the TPM under Owner authorization, and unwrapped blobs cannot be used without Owner authorisation. The TPM MUST NOT restrict the number of times that the contents of enc(DAA_tpmSpecific) or enc(v0) or enc(v1) can be used by the same combination of TPM and Owner that created them..

**Actions**

A Trusted Platform Module that receives a valid TPM_DAA_Sign command SHALL:

1. Use ownerAuth to verify that the Owner authorized all TPM_DAA_Sign input parameters.

**Stages**

0. If stage==0
   a. Set DAA_issuerSettings = inputData0
   b. Verify that all fields in DAA_issuerSettings are present and return error TPM_DAA_INPUT_DATA0 if not.
   c. set all fields in DAA_session = NULL
   d. Assign new handle for session
   e. Set outputData to new handle
   f. set DAA_session -> DAA_stage = 1

g. return TPM_SUCCESS

1. If stage==1
   a. Verify that DAA_session ->DAA_stage==1. Return TPM_DAA_STAGE and flush handle on mismatch
   b. Set DAA_tpmSpecific = unwrap(inputData0)
   c. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
   d. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific)
   e. obtain random data from the RNG and store it as DAA_session -> DAA_contextSeed
   f. set outputData = NULL
   g. set DAA_session -> DAA_stage =2
   h. return TPM_SUCCESS
2. If stage==2
   a. Verify that DAA_session ->DAA_stage==2. Return TPM_DAA_STAGE and flush handle on mismatch
   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
   c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch
   d. Set DAA_generic_R0 = inputData0
   e. Verify that SHA-1(DAA_generic_R0) == DAA_issuerSettings -> DAA_digest_R0 and return error TPM_DAA_INPUT_DATA0 on mismatch
   f. Set DAA_generic_n = inputData1
   g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1 on mismatch
   h. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and label them Y
   i. Set X = DAA_generic_R0
   j. Set n = DAA_generic_n
   k. Set DAA_session -> DAA_scratch = (X^Y) mod n
   l. set outputData = NULL
   m. increment DAA_session -> DAA_stage by 1
   n. return TPM_SUCCESS
3. If stage==3
   a. Verify that DAA_session ->DAA_stage==3. Return TPM_DAA_STAGE and flush handle on mismatch
   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
   c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch
   d. Set DAA_generic_R1 = inputData0
   e. Verify that SHA-1(DAA_generic_R1) == DAA_issuerSettings -> DAA_digest_R1 and return error TPM_DAA_INPUT_DATA0 on mismatch

f. Set DAA_generic_n = inputData1

g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1 on mismatch

h. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and label them Y

i. Set X = DAA_generic_R1

j. Set n = DAA_generic_n

k. Set Z = DAA_session -> DAA_scratch

l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n

m. set outputData = NULL

n. increment DAA_session -> DAA_stage by 1

o. return TPM_SUCCESS

4. If stage==4

a. Verify that DAA_session ->DAA_stage==4. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_generic_S0 = inputData0

e. Verify that SHA-1(DAA_generic_S0) == DAA_issuerSettings -> DAA_digest_S0 and return error TPM_DAA_INPUT_DATA0 on mismatch

f. Set DAA_generic_n = inputData1

g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1 on mismatch

h. obtain DAA_SIZE_r2 bits from MGF1("r2", DAA_session -> DAA_contextSeed), and label them Y

i. Set X = DAA_generic_S0

j. Set n = DAA_generic_n

k. Set Z = DAA_session -> DAA_scratch

l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n

m. set outputData = NULL

n. increment DAA_session -> DAA_stage by 1

o. return TPM_SUCCESS

5. If stage==5

a. Verify that DAA_session ->DAA_stage==5. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_generic_S1 = inputData0

e. Verify that SHA-1(DAA_generic_S1) == DAA_issuerSettings -> DAA_digest_S1 and return error TPM_DAA_INPUT_DATA0 on mismatch
f. Set DAA_generic_n = inputData1
g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and return error TPM_DAA_INPUT_DATA1 on mismatch
h. obtain DAA_SIZE_r4 bits from MGF1("r4", DAA_session -> DAA_contextSeed), and label them Y
i. Set X = DAA_generic_S1
j. Set n = DAA_generic_n
k. Set Z = DAA_session -> DAA_scratch
l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n
m. set outputData = DAA_session -> DAA_scratch
n. set DAA_session -> DAA_scratch = NULL
o. increment DAA_session -> DAA_stage by 1
p. return TPM_SUCCESS

6. If stage==6
   a. Verify that DAA_session ->DAA_stage==6. Return TPM_DAA_STAGE and flush handle on mismatch
   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
   c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch
   d. Set DAA_generic_gammma = inputData0
   e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings -> DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch
   f. Verify that inputSize1 == DAA_SIZE_w and return error TPM_DAA_INPUT_DATA1 on mismatch
   g. Set w = inputData1
   h. Set w1 = w^( DAA_issuerSettings -> DAA_generic_q) mod (DAA_generic_gamma)
   i. If w1 != 1 (unity), return error TPM_DAA_WRONG_W
   j. Set DAA_session -> DAA_scratch = w
   k. set outputData = NULL
   l. increment DAA_session -> DAA_stage by 1
   m. return TPM_SUCCESS.
7. If stage==7
   a. Verify that DAA_session ->DAA_stage==7. Return TPM_DAA_STAGE and flush handle on mismatch
   b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
   c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch
   d. Set DAA_generic_gamma = inputData0

e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings -> DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch

f. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0 ) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1 ) mod DAA_issuerSettings -> DAA_generic_q.

g. Set E = ((DAA_session -> DAA_scratch)^f) mod (DAA_generic_gamma).

h. Set outputData = E

i. increment DAA_session -> DAA_stage by 1

j. return TPM_SUCCESS.

8. If stage==8

a. Verify that DAA_session ->DAA_stage==8. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_generic_gamma = inputData0

e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings -> DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch

f. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and label them r0

g. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and label them r1

h. set r = r0 + 2^DAA_power0 * r1 mod (DAA_issuerSettings -> DAA_generic_q).

i. Set E1 = ((DAA_session -> DAA_scratch)^r) mod (DAA_generic_gamma)

j. Set DAA_session -> DAA_scratch = NULL

k. Set outputData = E1

l. increment DAA_session -> DAA_stage by 1

m. return TPM_SUCCESS.

9. If stage==9

a. Verify that DAA_session ->DAA_stage==9. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Verify that inputSize0 == sizeOf(TPM_DIGEST) and return error TPM_DAA_INPUT_DATA0 on mismatch

e. Set DAA_session -> DAA_digest = inputData0

f. obtain DAA_SIZE_NT bits from the RNG and label them NT

g. Set DAA_session -> DAA_digest to the SHA-1 ( DAA_session -> DAA_digest || NT )

h. Set outputData = NT

i. increment DAA_session -> DAA_stage by 1

j. return TPM_SUCCESS.

10. If stage==10
    a. Verify that DAA_session ->DAA_stage==10. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. Set selector = inputData0, verify that selector == 0 or 1, and return error TPM_DAA_INPUT_DATA0 on mismatch
    e. If selector == 1, verify that inputSize1 == sizeOf(TPM_DIGEST), and
    f. Set DAA_session -> DAA_digest to SHA-1 (DAA_session -> DAA_digest || 1 || inputData1)
    g. If selector == 0, verify that inputData1 is a handle to a TPM identity key (AIK), and
    h. Set DAA_session -> DAA_digest to SHA-1 (DAA_session -> DAA_digest || 0 || n2) where n2 is the modulus of the AIK
    i. Set outputData = DAA_session -> DAA_digest
    j. increment DAA_session -> DAA_stage by 1
    k. return TPM_SUCCESS.
11. If stage==11
    a. Verify that DAA_session ->DAA_stage==11. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. obtain DAA_SIZE_r0 bits from MGF1("r0", DAA_session -> DAA_contextSeed), and label them r0
    e. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0 ) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1 ) mod DAA_issuerSettings -> DAA_generic_q.
    f. Set f0 = f mod 2^DAA_power0 (erase all but the lowest DAA_power0 bits of f)
    g. Set s0 = r0 + (DAA_session -> DAA_digest)*(f0)
    h. set outputData = s0
    i. increment DAA_session -> DAA_stage by 1
    j. return TPM_SUCCESS
12. If stage==12
    a. Verify that DAA_session ->DAA_stage==12. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. obtain DAA_SIZE_r1 bits from MGF1("r1", DAA_session -> DAA_contextSeed), and label them r1

e. Set f = SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0 ) || SHA1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1 ) mod DAA_issuerSettings -> DAA_generic_q.

f. Shift f right by DAA_power0 bits (discard the lowest DAA_power0 bits) and label the result f1

g. Set s1 = r1 + (DAA_session -> DAA_digest)*(f1)

h. set outputData = s1

i. increment DAA_session -> DAA_stage by 1

j. return TPM_SUCCESS

13. If stage==13

a. Verify that DAA_session ->DAA_stage==13. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_private_v0= unwrap(inputData0)

e. Verify that SHA-1(DAA_private_v0) == DAA_tpmSpecific -> DAA_digest_v0 and return error TPM_DAA_INPUT_DATA0 on mismatch

f. obtain DAA_SIZE_r2 bits from MGF1("r2", DAA_session -> DAA_contextSeed), and label them r2

g. Set s2 = r2 + (DAA_session -> DAA_digest)*( DAA_private_v0) mod 2^DAA_power1 (erase all but the lowest DAA_power1 bits of s2)

h. Set DAA_session -> DAA_scratch = s2

i. set outputData = s2

j. increment DAA_session -> DAA_stage by 1

k. return TPM_SUCCESS

14. If stage==14

a. Verify that DAA_session ->DAA_stage==1. Return TPM_DAA_STAGE and flush handle on mismatch

b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch

c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_private_v0= unwrap(inputData0)

e. Verify that SHA-1(DAA_private_v0) == DAA_tpmSpecific -> DAA_digest_v0 and return error TPM_DAA_INPUT_DATA0 on mismatch

f. obtain DAA_SIZE_r2 bits from MGF1("r2", DAA_session -> DAA_contextSeed), and label them r2

g. Set s12 = r2 + (DAA_session -> DAA_digest)*(DAA_private_v0).

h. Shift s12 right by DAA_power1 bits (erase the lowest DAA_power1 bits).

i. Set DAA_session -> DAA_scratch = s12

j. set outputData = NULL

k. increment DAA_session -> DAA_stage by 1

l. return TPM_SUCCESS

15. If stage==15
    a. Verify that DAA_session ->DAA_stage==15. Return TPM_DAA_STAGE and flush handle on mismatch
    b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and return error TPM_DAA_ISSUER_SETTINGS on mismatch
    c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and return error TPM_DAA_TPM_SETTINGS on mismatch
    d. Set DAA_private_v1 = unwrap(inputData0)
    e. Verify that SHA-1(DAA_private_v1) == DAA_tpmSpecific -> DAA_digest_v1 and return error TPM_DAA_INPUT_DATA0 on mismatch
    f. obtain DAA_SIZE_r4 bits from MGF1("r4", DAA_session -> DAA_contextSeed), and label them r4
    g. Set s3 = r4 + (DAA_session -> DAA_digest)*(DAA_private_v1) + (DAA_session -> DAA_scratch).
    h. Set DAA_session -> DAA_scratch = NULL
    i. set outputData = s3
    j. increment DAA_session -> DAA_stage by 1
    k. return TPM_SUCCESS
16. If stage > 15, return error: TPM_DAA_STAGE

# 27. GPIO

## 27.1 TPM_GPIO_AuthChannel

***Start of informative comment:***

This command authorizes later use of an IO channel under restricted conditions.

This command builds an authorization block that is later loaded into the TPM for the TPM_GPIO_ReadWrite or TPM_SetRedirection command. This authorization block includes both an HMAC integrity digest using tpmProof as the key and an encrypted authorization value if appropriate.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_GPIO_AuthChannel |
| 4 | 20 | 2S | 20 | TPM_ENCAUTH | ioAuth | The encrypted authorization data for the sealed data. The encryption key is the shared secret from the OSAP protocol. |
| 5 | 4 | 3S | 4 | UINT32 | sizeChannel | Size in bytes of channel |
| 6 | <> | 4S | <> | TPM_GPIO_CHANNEL | channel | Mode for the command |
| 7 | 4 | | | TPM_AUTHHANDLE | ownerAuthHandle | The authorization handle used for the TPM Owner. |
| | | 2 H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3 H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with ownerAuthHandle |
| 9 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag, ignored. |
| 10 | 20 | | | TPM_AUTHDATA | ownerAuth | Authorization. HMAC key: tpmOwnerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal TPM_ORD_GPIO_AuthChannel |
| 4 | <> | 3S | <> | TPM_GPIO_AUTHORIZE | channelAuth | Computed authorization structure for this channel. |
| 4 | 20 | 2 H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with ownerAuthHandle |
| 5 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Always FALSE. |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | Authorization HMAC key: tpmOwnerAuth. |

**Actions**

1. Validate owner authorization for the input parameters, return TPM_AUTHFAIL on error
2. Validate that the channel description is consistent
   a. Consistency includes busInfo, ps, channelNumber and attr.
   b. Validate that the read or write operations described in channel->attributes are legal for this IO capability
3. Map G1 to a TPM_GPIO_AUTHORIZE structure
4. Copy channel to G1->channel
5. If channel -> attr -> TPM_GPIO_ATTR_AUTH is TRUE
   a. Verify that ownerAuthHandle points to an OSAP authorization session
   b. Create s1 string by concatenating (ownerAuth -> shared secret || authLastNonceEven)
   c. Create x1 by performing a SHA1 hash of s1
   d. Fill in G1 -> sensitiveData->authData with the XOR of x1 and ioAuth
   e. Set G1 -> additionalData to the appropriate IV
6. Else
   a. Set G1 -> sensitiveDataSize to 0 and G1 -> additionalDataSize to 0
7. Set G1 -> integrityDigest to NULL
8. Set G1 -> integrityDigest to the HMAC of G1, using tpmProof as the key.
9. If G1 -> sensitiveDataSize is not 0
   a. Encrypt G1->sensitiveData using the tpm encryption key

## 27.2 TPM_GPIO_ReadWrite

***Start of informative comment:***

This command reads and or writes to an IO channel.

The authorization to use this channel is either contained in the PCR state and/or Locality, or is controlled by an authorization value encrypted and contained in the authorization input.

This command is not owner authorized. The generation of the authorization package requires owner authorization and the owner controls the use of the IO channel by controlling access to the stored authorization value.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_GPIO_ReadWrite |
| 4 | 4 | 2S | 4 | UINT32 | channelAuthSize | Size in bytes of channelAuth |
| 5 | <> | 3S | <> | TPM_GPIO_AUTHORIZE | channelAuth | Channel information and authorization |
| 6 | 4 | 4S | 4 | UINT32 | readBytes | Number of bytes to be read |
| 7 | 4 | 5S | 4 | UINT32 | writeBytes | Number of bytes to be wriiten; size of writeData |
| 8 | <> | 6S | <> | BYTE [ ] | writeData | The bytes to be written to the channel |
| 9 | 4 | | | TPM_AUTHHANDLE | ownerAuthHandle | The authorization handle used for the TPM Owner. |
| | | 2 H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 10 | 20 | 3 H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with ownerAuthHandle |
| 11 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag. |
| 12 | 20 | | | TPM_AUTHDATA | ownerAuth | Authorization HMAC key channel auth -> sensitive data |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation |
| 4 | 4 | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal TPM_ORD_GPIO_ReadWrite |
| 5 | 4 | 3S | 4 | UINT32 | readDataSize | Number of bytes read from the channel; size of readData |
| 6 | <> | 4S | <> | BYTE[ ] | readData | Data returned from the channel |
| 7 | 20 | 2 H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3 H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with ownerAuthHandle |
| 8 | 1 | 4 H1 | 1 | BOOL | continueAuthSession | Continue use flag. |
| 9 | 20 | | | TPM_AUTHDATA | resAuth | Authorization HMAC key channel auth -> sensitive data |

**Actions**

1. Validate that channelAuth->channel->busInfo and channelAuth->channel->channelNum refer to a valid IO capability on this TPM
2. Validate that the read or write operations described in channelAuth->channel->attributes are legal for this IO capability.
3. If readBytes is non-zero and channel->attributes -> TPM_GPIO_ATTR_READ is clear, return error.
4. If writeBytes is non-zero and channel->attributes -> TPM_GPIO_ATTR_WRITE is clear, return error.
5. If channelAuth->channel->attributes -> PHYSPRES is set, verify that physical presence is asserted
6. If channelAuth -> channel -> attributes -> TPM_GPIO_ATTR_REDIR_KEY or channelAuth -> channel-> attributes -> TPM_GPIO_ATTR_REDIR is set, return error.
7. If channel is connected to PCRs or localities, verify that they have the correct value.
8. If channelAuth->channel->attributes.TPM_GPIO_ATTR_AUTH is set
   a. If tag is not TPM_TAG_RQU_AUTH1_COMMAND return error.
   b. If channelAuth->additionalDataSize or channelAuth->sensitiveDataSize are 0 return error.
   c. Decrypt channelAuth->sensitiveData, compute the input authorization HMAC using the decrypted authorization value as the HMAC key.
   d. Return error if mismatch.
9. Else
   a. If TAG is not TPM_TAG_RQU_COMMAND return error.
10. Set A1 to channelAuth->integrityDigest.
11. Set channelAuth->integrityDigest to NULL, compute A2, the HMAC of channelAuth using tpmProof as the key.
12. Compare A1 with A2, return error on mismatch
13. Perform read and/or write as requested to the channel and address specified in channel. If both a read and write are specified, the write occurs before the read.

# 28. Deprecated commands

***Start of informative comment:***

This section covers the commands that were in version 1.1 but now have new functionality in other functions. The deprecated commands are still available in 1.2 but all new software should use the new functionality.

There is no requirement that the deprecated commands work with new structures.

***End of informative comment.***

1. Commands deprecated in version 1.2 MUST work with version 1.1 structures
2. Commands deprecated in version 1.2 MAY work with version 1.2 structures

# 28.1 Key commands

***Start of informative comment:***

The key commands are deprecated as the new way to handle keys is to use the standard context commands. So TPM_EvictKey is now handled by TPM_FlushSpecific, TPM_TerminateHandle by TPM_FlushSpecific.

***End of informative comment.***

## 28.1.1 TPM_EvictKey

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_EvictKey |
| 4 | 4 | | | TPM_KEY_HANDLE | evictHandle | The handle of the key to be evicted. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Actions**

The TPM will invalidate the key stored in the specified handle and return the space to the available internal pool for subsequent query by TPM_GetCapability and usage by TPM_LoadKey. If the specified key handle does not correspond to a valid key, an error will be returned.

### 28.1.2 TPM_Terminate_Handle

***Start of informative comment:***

This allows the TPM manager to clear out information in a session handle.

The TPM may maintain the authorization session even though a key attached to it has been unloaded or the authorization session itself has been unloaded in some way. When a command is executed that requires this session, it is the responsibility of the external software to load both the entity and the authorization session information prior to command execution.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Terminate_Handle. |
| 4 | 4 | | | TPM_AUTHHANDLE | handle | The handle to terminate |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Descriptions**

The TPM SHALL terminate the session and destroy all data associated with the session indicated.

**Actions**

A TPM SHALL unilaterally perform the actions of TPM_Terminate_Handle upon detection of the following events:

1. Completion of a received command whose authorization “continueUse” flag is FALSE.
2. Completion of a received command when a shared secret derived from the authorization session was exclusive-or’ed with data (to provide confidentiality for that data). This occurs during execution of a TPM_ChangeAuth command, for example.
3. When the associated entity is destroyed (in the case of TPM Owner or SRK, for example)
4. Upon execution of TPM_Init
5. When the command returns an error. This is due to the fact that when returning an error the TPM does not send back nonceEven. There is no way to maintain the rolling nonces, hence the TPM MUST terminate the authorization session.
6. Failure of an authorization check belonging to that authorization session.

# 28.2 Context management

***Start of informative comment:***

The 1.1 context commands were written for specific resource types. The 1.2 commands are generic for all resource types. So the Savexxx commands are replaced by TPM_SaveContext and the LoadXXX commands by TPM_LoadContext.

***End of informative comment.***

## 28.2.1 TPM_SaveKeyContext

***Start of informative comment:***

SaveKeyContext saves a loaded key outside the TPM. After creation of the key context blob the TPM automatically releases the internal memory used by that key. The format of the key context blob is specific to a TPM.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SaveKeyContext |
| 4 | 4 | | | TPM_KEY_HANDLE | keyHandle | The key which will be kept outside the TPM |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 4 | | | UINT32 | keyContextSize | The actual size of the outgoing key context blob. If the command fails the value will be 0 |
| 5 | <> | | | BYTE[] | keyContextBlob | The key context blob. |

**Description**

1. This command allows saving a loaded key outside the TPM. After creation of the KeyContextBlob, the TPM automatically releases the internal memory used by that key. The format of the key context blob is specific to a TPM.
2. A TPM protected capability belonging to the TPM that created a key context blob MUST be the only entity that can interpret the contents of that blob. If a cryptographic technique is used for this purpose, the level of security provided by that technique SHALL be at least as secure as a 2048 bit RSA algorithm. Any secrets (such as keys) used in such a cryptographic technique MUST be generated using the TPM's random number generator. Any symmetric key MUST be used within the power-on session during which it was created, only.

3. A key context blob SHALL enable verification of the integrity of the contents of the blob by a TPM protected capability.
4. A key context blob SHALL enable verification of the session validity of the contents of the blob by a TPM protected capability. The method SHALL ensure that all key context blobs are rendered invalid if power to the TPM is interrupted.

### 28.2.2 TPM_LoadKeyContext

***Start of informative comment:***

LoadKeyContext loads a key context blob into the TPM previously retrieved by a SaveKeyContext call. After successful completion the handle returned by this command can be used to access the key.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_LoadKeyContext |
| 4 | 4 | | | UINT32 | keyContextSize | The size of the following key context blob. |
| 5 | <> | | | BYTE[] | keyContextBlob | The key context blob. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 4 | | | TPM_KEY_HANDLE | keyHandle | The handle assigned to the key after it has been successfully loaded. |

**Description**

1. This command allows loading a key context blob into the TPM previously retrieved by a TPM_SaveKeyContext call. After successful completion the handle returned by this command can be used to access the key.
2. The contents of a key context blob SHALL be discarded unless the contents have passed an integrity test. This test SHALL (statistically) prove that the contents of the blob are the same as when the blob was created.
3. The contents of a key context blob SHALL be discarded unless the contents have passed a session validity test. This test SHALL (statistically) prove that the blob was created by this TPM during this power-on session.

### 28.2.3 TPM_SaveAuthContext

***Start of informative comment:***

SaveAuthContext saves a loaded authorization session outside the TPM. After creation of the authorization context blob, the TPM automatically releases the internal memory used by that session. The format of the authorization context blob is specific to a TPM.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_SaveAuthContext |
| 4 | 4 | | | TPM_AUTHHANDLE | authandle | Authorization session which will be kept outside the TPM |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 4 | | | UINT32 | authContextSize | The actual size of the outgoing authorization context blob. If the command fails the value will be 0. |
| 5 | <> | | | BYTE[] | authContextBlob | The authorization context blob. |

**Description**

This command allows saving a loaded authorization session outside the TPM. After creation of the authContextBlob, the TPM automatically releases the internal memory used by that session. The format of the authorization context blob is specific to a TPM.

A TPM protected capability belonging to the TPM that created an authorization context blob MUST be the only entity that can interpret the contents of that blob. If a cryptographic technique is used for this purpose, the level of security provided by that technique SHALL be at least as secure as a 2048 bit RSA algorithm. Any secrets (such as keys) used in such a cryptographic technique MUST be generated using the TPM's random number generator. Any symmetric key MUST be used within the power-on session during which it was created, only.

An authorization context blob SHALL enable verification of the integrity of the contents of the blob by a TPM protected capability.

An authorization context blob SHALL enable verification of the session validity of the contents of the blob by a TPM protected capability. The method SHALL ensure that all authorization context blobs are rendered invalid if power to the TPM is interrupted.

### 28.2.4 TPM_LoadAuthContext

***Start of informative comment:***

LoadAuthContext loads an authorization context blob into the TPM previously retrieved by a SaveAuthContext call. After successful completion the handle returned by this command can be used to access the authorization session.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_LoadAuthContext |
| 4 | 4 | | | UINT32 | authContextSize | The size of the following authorization context blob. |
| 5 | <> | | | BYTE[] | authContextBlob | The authorization context blob. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 4 | | | TPM_KEY_HANDLE | authHandle | The handle assigned to the authorization session after it has been successfully loaded. |

**Description**

This command allows loading an authorization context blob into the TPM previously retrieved by a TPM_SaveAuthContext call. After successful completion the handle returned by this command can be used to access the authorization session.

The contents of an authorization context blob SHALL be discarded unless the contents have passed an integrity test. This test SHALL (statistically) prove that the contents of the blob are the same as when the blob was created.

The contents of an authorization context blob SHALL be discarded unless the contents have passed a session validity test. This test SHALL (statistically) prove that the blob was created by this TPM during this power-on session.

# 28.3 DIR commands

***Start of informative comment:***

The DIR commands are replaced by the NV storage commands.

The DIR in 1.1 is now index 1 of the NV storage area and is always available for the TPM to use.

***End of informative comment.***

## 28.3.1 TPM_DirWriteAuth

***Start of informative comment:***

The TPM_DirWriteAuth operation provides write access to the Data Integrity Registers. DIRs are non-volatile memory registers held in a TPM-shielded location. Owner authentication is required to authorize this action. Version 1 requires only one DIR. If the DIR named does not exist, the TPM_DirRead operation returns TPM_BADINDEX.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_DirWriteAuth. |
| 4 | 4 | 2S | 4 | TPM_DIRINDEX | dirIndex | Index of the DIR |
| 5 | 20 | 3S | 20 | TPM_DIRVALUE | newContents | New value to be stored in named DIR |
| 6 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for command. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 7 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 9 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. See section 4.3. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_DirWriteAuth |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Actions**

1. Validate that authHandle contains a TPM Owner authorization to excute the TPM_DirWriteAuth command
2. Validate that dirIndex points to a valid DIR on this TPM
3. Write newContents into the DIR pointed to by dirIndex

### 28.3.2 TPM_DirRead

***Start of informative comment:***

The TPM_DirRead operation provides read access to the DIRs. No authentication is required to perform this action because typically no cryptographically useful authorization data is available early in boot. TSS implementors may choose to provide other means of authorizing this action. Version 1 requires only one DIR. If the DIR named does not exist, the TPM_DirRead operation returns TPM_BADINDEX.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_DirRead. |
| 4 | 4 | | | TPM_DIRINDEX | dirIndex | Index of the DIR to be read |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 20 | | | TPM_DIRVALUE | dirContents | The current contents of the named DIR |

**Actions**

1. Validate that dirIndex points to a valid DIR on this TPM
2. Return the contents of the DIR in dirContents

# 28.4 Change Auth

***Start of informative comment:***

The change auth commands can be duplicated by creating a transport session with confidentiality and issuing the changeAuth command.

***End of informative comment.***

## 28.4.1 TPM_ChangeAuthAsymStart

***Start of informative comment:***

The TPM_ChangeAuthAsymStart starts the process of changing authorization for an entity. It sets up an OIAP session that must be retained for use by its twin TPM_ChangeAuthAsymFinish command.

TPM_ChangeAuthAsymStart creates a temporary asymmetric public key "tempkey" to provide confidentiality for new authorization data to be sent to the TPM. TPM_ChangeAuthAsymStart certifies that tempkey was generated by a genuine TPM, by generating a certifyInfo structure that is signed by a TPM identity. The owner of that TPM identity must cooperate to produce this command, because TPM_ChangeAuthAsymStart requires authorization to use that identity.

It is envisaged that tempkey and certifyInfo are given to the owner of the entity whose authorization is to be changed. That owner uses certifyInfo and a TPM_IDENTITY_CREDENTIAL to verify that tempkey was generated by a genuine TPM. This is done by verifying the TPM_IDENTITY_CREDENTIAL using the public key of a CA, verifying the signature on the certifyInfo structure with the public key of the identity in TPM_IDENTITY_CREDENTIAL, and verifying tempkey by comparing its digest with the value inside certifyInfo. The owner uses tempkey to encrypt the desired new authorization data and inserts that encrypted data in a TPM_ChangeAuthAsymFinish command, in the knowledge that only a TPM with a specific identity can interpret the new authorization data.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ChangeAuthAsymStart. |
| 4 | 4 | | | TPM_KEY_HANDLE | idHandle | The keyHandle identifier of a loaded identity ID key |
| 5 | 20 | 2s | 20 | TPM_NONCE | antiReplay | The nonce to be inserted into the certifyInfo structure |
| 6 | <> | 3S | <> | TPM_KEY_PARMS | tempKey | Structure contains all parameters of ephemeral key. |
| 7 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for idHandle authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | 20 | | | TPM_AUTHDATA | idAuth | Authorization. HMAC key: idKey.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ChangeAuthAsymStart |
| 7 | 95 | 3S | 95 | TPM_CERTIFY_INFO | certifyInfo | The certifyInfo structure that is to be signed. |
| 8 | 4 | 4S | 4 | UINT32 | sigSize | The used size of the output area for the signature |
| 9 | <> | 5S | <> | BYTE[ ] | sig | The signature of the certifyInfo parameter. |
| 10 | 4 | 6s | 4 | TPM_KEY_HANDLE | ephHandle | The keyHandle identifier to be used by ChangeAuthAsymFinish for the ephemeral key |
| 11 | <> | 7S | <> | TPM_KEY | tempKey | Structure containing all parameters and public part of ephemeral key. TPM_KEY.encSize is set to 0. |
| 12 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 13 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 14 | 20 | | | TPM_AUTHDATA | resAuth | Authorization. HMAC key: idKey.usageAuth. |

**Actions**

1. The TPM SHALL verify the authorization to use the TPM identity key held in idHandle. The TPM MUST verify that the key is a TPM identity key.
2. The TPM SHALL validate the algorithm parameters for the key to create from the tempKey parameter.
3. Recommended key type is RSA
4. Minimum RSA key size MUST is 512 bits, recommended RSA key size is 1024
5. For other key types the minimum key size strength MUST be comparable to RSA 512
6. If the TPM is not designed to create a key of the requested type, return the error code TPM_BAD_KEY_PROPERTY
7. The TPM SHALL create a new key (k1) in accordance with the algorithm parameter. The newly created key is pointed to by ephHandle.
8. The TPM SHALL fill in all fields in tempKey using k1 for the information. The TPM_KEY -> encSize MUST be 0.
9. The TPM SHALL fill in certifyInfo using k1 for the information. The certifyInfo -> data field is supplied by the antiReplay.
10. The TPM then signs the certifyInfo parameter using the key pointed to by idHandle. The resulting signed blob is returned in sig parameter

**Field Descriptions for certifyInfo parameter**

| Type | Name | Description |
|---|---|---|
| TPM_VERSION | Version | TPM version structure; section TODOREF |
| keyFlags | Redirection | This SHALL be set to FALSE |
| | Migratable | This SHALL be set to FALSE |
| | Volatile | This SHALL be set to TRUE |
| TPM_AUTH_DATA_USAGE | authDataUsage | This SHALL be set to TPM_AUTH_NEVER |
| TPM_KEY_USAGE | KeyUsage | This SHALL be set to TPM_KEY_AUTHCHANGE |
| UINT32 | PCRInfoSize | This SHALL be set to 0 |
| TPM_DIGEST | pubDigest | This SHALL be the hash of the public key being certified. |
| TPM_NONCE | Data | This SHALL be set to antiReplay |
| TPM_KEY_PARMS | info | This specifies the type of key and its parameters. |
| BOOL | parentPCRStatus | This SHALL be set to FALSE. |

## 28.4.2 TPM_ChangeAuthAsymFinish

***Start of informative comment:***

The TPM_ChangeAuth command allows the owner of an entity to change the authorization data for the entity.

The command requires the cooperation of the owner of the parent of the entity, since authorization must be provided to use that parent entity. The command requires knowledge of the existing authorization information and passes the new authorization information. The newAuthLink parameter proves knowledge of existing authorization information and new authorization information. The new authorization information "encNewAuth" is encrypted using the "tempKey" variable obtained via TPM_ChangeAuthAsymStart.

A parent therefore retains control over a change in the authorization of a child, but is prevented from knowing the new authorization data for that child.

The changeProof parameter provides a proof that the new authorization value was properly inserted into the entity. The inclusion of a nonce from the TPM provides an entropy source in the case where the authorization value may be in itself be a low entropy value (hash of a password etc).

***End of informative comment***.

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ChangeAuthAsymFinish |
| 4 | 4 | | | TPM_KEY_HANDLE | parentHandle | The keyHandle of the parent key for the input data |
| 5 | 4 | | | TPM_KEY_HANDLE | ephHandle | The keyHandle identifier for the ephemeral key |
| 6 | 2 | 3S | 2 | TPM_ENTITY_TYPE | entityType | The type of entity to be modified |
| 7 | 20 | 4s | 20 | TPM_HMAC | newAuthLink | HMAC calculation that links the old and new authorization values together |
| 8 | 4 | 5S | 4 | UINT32 | newAuthSize | Size of encNewAuth |
| 9 | <> | 6S | <> | BYTE[ ] | encNewAuth | New authorization data encrypted with ephemeral key. |
| 10 | 4 | 7S | 4 | UINT32 | encDataSize | The size of the inData parameter |
| 11 | <> | 8S | <> | BYTE[ ] | encData | The encrypted entity that is to be modified. |
| 12 | 4 | | | TPM_AUTHHANDLE | authHandle | Authorization for parent key. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 13 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 14 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 15 | 20 | | | TPM_AUTHDATA | privAuth | The authorization digest for inputs and parentHandle. HMAC key: parentKey.usageAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |

| | | | | | | |
|---|---|---|---|---|---|---|
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_ChangeAuthAsymFinish |
| 4 | 4 | 3S | 4 | UINT32 | outDataSize | The used size of the output area for outData |
| 5 | <> | 4S | <> | BYTE[ ] | outData | The modified, encrypted entity. |
| 6 | 20 | 5s | 20 | TPM_NONCE | saltNonce | A nonce value from the TPM RNG to add entropy to the changeProof value |
| 7 | <> | 6S | <> | TPM_DIGEST | changeProof | Proof that authorization data has changed. |
| 8 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 10 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: parentKey.usageAuth. |

**Description**

If the parentHandle points to the SRK then the HMAC key MUST be built using the TPM Owner authorization.

**Actions**

1. The TPM SHALL validate that the authHandle parameter authorizes use of the key in parentHandle.
2. The encData field MUST be the encData field from TPM_STORED_DATA or TPM_KEY.
3. The TPM SHALL create e1 by decrypting the entity held in the encData parameter.
4. The TPM SHALL create a1 by decrypting encNewAuth using the authHandle -> TPM_KEY_AUTHCHANGE private key. a1 is a structure of type TPM_CHANGEAUTH_VALIDATE.
5. The TPM SHALL create b1 by performing the following HMAC calculation: b1 = HMAC (a1 -> newAuthSecret). The secret for this calculation is encData -> currentAuth. This means that b1 is a value built from the current authorization value (encData -> currentAuth) and the new authorization value (a1 -> newAuthSecret).
6. The TPM SHALL compare b1 with newAuthLink. The TPM SHALL indicate a failure if the values do not match.
7. The TPM SHALL replace e1 -> authData with a1 -> newAuthSecret
8. The TPM SHALL encrypt e1 using the appropriate functions for the entity type. The key to encrypt with is parentHandle.
9. The TPM SHALL create saltNonce by taking the next 20 bytes from the TPM RNG.
10. The TPM SHALL create changeProof a HMAC of (saltNonce concatenated with a1 -> n1) using a1 -> newAuthSecret as the HMAC secret.
11. The TPM MUST destroy the TPM_KEY_AUTHCHANGE key associated with the authorization session.

# 28.5 TPM_Reset

***Start of informative comment:***

TPM_Reset releases all resources associated with existing authorization sessions. This is useful if a TSS driver has lost track of the state in the TPM.

***End of informative comment.***

**Deprecated Command in 1.2**

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal, fixed value of TPM_ORD_Reset. |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |

**Description**

This is a deprecated command in V1.2. This command in 1.1 only referenced authorization sessions and is not upgraded to affect any other TPM entity in 1.2

**Actions**

1. The TPM invalidates all resources allocated to authorization sessions as per version 1.1 extant in the TPM
   a. This includes structures created by TPM_AuthContextSave and TPM_KeyContextSave
   b. Structures created by TPM_Contextxxx (the new 1.2 commands) are not affected by this command
2. The TPM does not reset any PCR or DIR values.
3. The TPM does not reset any flags in the TPM_STCLEAR_FLAGS structure.
4. The TPM does not reset or invalidate any keys

# 28.6 TPM_CertifySelfTest

***Start of informative comment:***

CertifySelfTest causes the TPM to perform a full self-test and return an authenticated value if the test passes.

If a caller itself requires proof, it is sufficient to use any signing key for which only the TPM and the caller have authorization data.

If a caller requires proof for a third party, the signing key must be one whose signature is trusted by the third party. A TPM-identity key may be suitable.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CertifySelfTest |
| 4 | 4 | | | TPM_KEY_HANDLE | keyHandle | The keyHandle identifier of a loaded key that can perform digital signatures. |
| 5 | 20 | 2S | 20 | TPM_NONCE | antiReplay | AnitReplay nonce to prevent replay of messages |
| 6 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 7 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 9 | 20 | | | TPM_AUTHDATA | privAuth | The authorization digest that authorizes the inputs and use of keyHandle. HMAC key: key.usageAuth |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_CertifySelfTest |
| 4 | 4 | 3S | 4 | UINT32 | sigSize | The length of the returned digital signature |
| 5 | <> | 4S | <> | BYTE[ ] | sig | The resulting digital signature. |
| 6 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 7 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 8 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth |

**Description**

The key in keyHandle MUST have a KEYUSAGE value of type TPM_KEY_SIGNING or TPM_KEY_LEGACY or TPM_KEY_IDENTITY.

Information returned by TPM_CertifySelfTest MUST NOT aid identification of an individual TPM.

**Actions**

1. The TPM SHALL perform TPM_SelfTestFull. If the test fails the TPM returns the appropriate error code.
2. After successful completion of the self-test the TPM then validates the authorization to use the key pointed to by keyHandle
3. Create t1 the null terminated string of “Test Passed”
4. The TPM creates m2 the message to sign by concatenating t1 || AntiReplay || ordinal.
5. The TPM signs m2 using the key identified by keyHandle, and returns the signature as sig.

# 28.7 TPM_OwnerReadPubek

***Start of informative comment:***

Return the endorsement key public portion. This is authorized by the TPM Owner

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_OwnerReadPubek |
| 4 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 5 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 7 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: ownerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_OwnerReadPubek |
| 4 | <> | 3S | <> | TPM_PUBKEY | pubEndorsementKey | The public endorsement key |
| 5 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 7 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: ownerAuth. |

**Description**

This command returns the PUBEK.

**Actions**

The TPM_ReadPubek command SHALL

1. Validate the TPM Owner authorization to execute this command
2. Export the PUBEK

# 29. Deleted Commands

***Start of informative comment:***

These commands are no longer active commands. Their removal is due to security concerns with their use.

***End of informative comment.***

1. The TPM MUST return TPM_BAD_ORDINAL for any deleted command

# 29.1 TPM_GetCapabilityOwner

***Start of informative comment:***

This is broke in 1.1. It can provide information to TPM_GetCapabilitySigned which may result in a invalid signature. As such the command is deleted and not deprecated.

TPM_GetCapabilityOwner enables the TPM Owner to retrieve all the non-volatile flags and the volatile flags in a single operation.

The flags summarize many operational aspects of the TPM. The information represented by some flags is private to the TPM Owner. So, for simplicity, proof of ownership of the TPM must be presented to retrieve the set of flags. When necessary, the flags that are not private to the Owner can be deduced by Users via other (more specific) means.

The normal TPM authorization mechanisms are sufficient to prove the integrity of the response. No additional integrity check is required.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_GetCapbilityOwner |
| 4 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for Owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 5 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 6 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 7 | 20 | | | TPM_AUTHDATA | ownerAuth | The authorization digest for inputs and owner authorization. HMAC key: OwnerAuth. |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. See section 4.3. |
| 4 | 4 | 2S | 4 | TPM_VERSION | version | A properly filled out version structure. |
| 5 | 4 | 3S | 4 | UINT32 | non_volatile_flags | The current state of the non-volatile flags. |
| 6 | 4 | 4S | 4 | UINT32 | volatile_flags | The current state of the volatile flags. |
| 7 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 9 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: OwnerAuth. |

**Description**

For 31>=N>=0

1. Bit-N of the TPM_PERMANENT_FLAGS structure is the Nth bit after the opening bracket in the definition of TPM_PERMANENT_FLAGS in the version of the specification indicated by the parameter "version". The bit immediately after the opening bracket is the $0^{th}$ bit.
2. Bit-N of the TPM_STCLEAR_FLAGS structure is the Nth bit after the opening bracket in the definition of TPM_STCLEAR_FLAGS in the version of the specification indicated by the parameter "version". The bit immediately after the opening bracket is the $0^{th}$ bit.
3. Bit-N of non_volatile_flags corresponds to the Nth bit in TPM_PERMANENT_FLAGS, and the lsb of non_volatile_flags corresponds to bit0 of TPM_PERMANENT_FLAGS
4. Bit-N of volatile_flags corresponds to the Nth bit in TPM_STCLEAR_FLAGS, and the lsb of volatile_flags corresponds to bit0 of TPM_STCLEAR_FLAGS

**Actions**

1. The TPM validates that the TPM Owner authorizes the command.
2. The TPM creates the parameter non_volatile_flags by setting each bit to the same state as the corresponding bit in TPM_PERMANENT_FLAGS. Bits in non_volatile_flags for which there is no corresponding bit in TPM_PERMANENT_FLAGS are set to zero.
3. The TPM creates the parameter volatile_flags by setting each bit to the same state as the corresponding bit in TPM_STCLEAR_FLAGS. Bits in volatile_flags for which there is no corresponding bit in TPM_STCLEAR_FLAGS are set to zero.
4. The TPM generates the parameter "version".
5. The TPM returns non_volatile_flags, volatile_flags and version to the caller.

# 29.2 TPM_GetCapabilitySigned

***Start of informative comment:***

TPM_GetCapabilitySigned is almost the same as TPM_GetCapability. The differences are that the input includes a challenge (a nonce) and the response includes a digital signature to vouch for the source of the answer.

If a caller itself requires proof, it is sufficient to use any signing key for which only the TPM and the caller have authorization data.

If a caller requires proof for a third party, the signing key must be one whose signature is trusted by the third party. A TPM-identity key may be suitable.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_GetCapabilitySigned |
| 4 | 4 | | | TPM_KEY_HANDLE | keyHandle | The handle of a loaded key that can perform digital signatures. |
| 5 | 20 | 2s | 20 | TPM_NONCE | antiReplay | Nonce provided to allow caller to defend against replay of messages |
| 6 | 4 | 3s | 4 | TPM_CAPABILITY_AREA | capArea | Partition of capabilities to be interrogated |
| 7 | 4 | 4s | 4 | UINT32 | subCapSize | Size of subCap parameter |
| 8 | <> | 5s | <> | BYTE[] | subCap | Further definition of information |
| 9 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for keyHandle authorization |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 10 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 11 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 12 | 20 | | | TPM_AUTHDATA | privAuth | The authorization digest that authorizes the use of keyHandle. HMAC key: key.usageAuth |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_GetCapabilitySigned |
| 4 | 4 | 3S | 4 | TPM_VERSION | version | A properly filled out version structure. |
| 5 | 4 | 4S | 4 | UINT32 | respSize | The length of the returned capability response |
| 6 | <> | 5S | <> | BYTE[ ] | resp | The capability response |
| 7 | 4 | 6S | 4 | UINT32 | sigSize | The length of the returned digital signature |

| | | | | | | |
|---|---|---|---|---|---|---|
| 8 | <> | 7S | <> | BYTE[ ] | sig | The resulting digital signature. |
| 9 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 10 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 11 | 20 | | | TPM_AUTHDATA | resAuth | The authorization digest for the returned parameters. HMAC key: key.usageAuth |

**Description**

The key in keyHandle MUST have a KEYUSAGE value of type TPM_KEY_SIGNING or TPM_KEY_LEGACY or TPM_KEY_IDENTITY.

**Actions**

1. The TPM validates the authority to use keyHandle
2. The TPM calls TPM_GetCapability passing the capArea and subCap fields and saving the resp field as R1
3. The TPM creates S1 by taking a SHA1 hash of the concatenation (r1 || antiReplay).
4. The TPM validates the authority to use keyHandle
5. The TPM creates a digital signature of S1 using the key in keyHandle and returns the result in sig.

## 29.3 TPM_GetOrdinalAuditStatus

***Start of informative comment:***

Get the status of the audit flag for the given ordinal.

***End of informative comment.***

**Incoming Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_GetOrdinalAuditStatus |
| 4 | 4 | | | TPM_COMMAND_CODE | ordinalToQuery | The ordinal whose audit flag is to be queried |

**Outgoing Operands and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 4 | 1 | | | BOOL | State | Value of audit flag for ordinalToQuery |

**Actions**

1. The TPM returns the Boolean value for the given ordinal. The value is TRUE if the command is being audited.

## 29.4 Audit Generation

***Start of informative comment:***

The TPM generates an audit event in response to the TPM executing a function that has the audit flag set to TRUE for that function.

The TPM maintains an extended value for all audited operations.

Input audit generation occurs before the listed actions and output audit generation occurs after the listed actions.

***End of informative comment.***

**Description**

The TPM extends the audit digest whenever the ordinalAuditStatus is TRUE for the ordinal about to be executed.

**Actions**

The TPM will execute the ordinal and perform auditing in the following manner

7. Map V1 to TPM_STANY_DATA

8. Map P1 to TPM_PERMANENT_DATA
9. If V1 -> auditDigest is NULL
   a. Increment P1 -> auditMonotonicCounter by 1
10. Create A1 a TPM_AUDIT_EVENT_IN structure
   a. Set A1 -> inputParms to the input parameters from the command
   b. Set A1 -> ordinal to the ordinal of the command
   c. Set A1 -> auditCount to P1 -> auditMonotonicCounter
   d. Set V1 -> auditDigest to SHA-1 (V1 -> auditDigest || A1)
11. Execute command
   a. Execution implies the performance of the listed actions for the ordinal.
12. Create A2 a TPM_AUDIT_EVENT_OUT structure
   a. Set A2 -> outputParms to the output parameters from the command
   b. Set A2 -> returnCode to the return code for the command
   c. Set A2 -> ordinal to the ordinal of the command
   d. Set A2 -> auditCount to P1 -> auditMonotonicCounter
   e. Set V1 -> auditDigest to SHA-1 (V1 -> auditDigest || A2)

## 29.5 Effect of audit failing after successful completion of a command

***Start of informative comment:***

An operation could complete successfully and then when the TPM attempts to audit the command the audit process could have an internal error that forces the TPM to return an error.

The TPM is unable to return the results of the command that ran and this includes success or failure. To indicate to the caller the TPM will one of two error codes TPM_AUDITFAIL_SUCCESSFUL and TPM_AUDITFAIL_UNSUCCESSFUL. These two error codes indicate if the command succeeded or failed. The purpose of this command is to indicate to the caller what occurred with the command execution.

This is new functionality that changes the 1.1 TPM functionality when this condition occurs.

***End of informative comment.***

4. When after successful completion of an operation, and in performing the audit process, the TPM has an internal failure (unable to write, SHA-1 failure etc.) the TPM MUST set the internal TPM state such that the TPM returns the TPM_FAILEDSELFTEST error.
5. If the command is returning a return code that indicates successful execution of the command the TPM SHALL change the return code to TPM_AUDITFAIL_SUCCESSFUL. For all other error codes the TPM MUST return TPM_AUDITFAIL_UNSUCCESSFUL.
6. If the TPM is permanently nonrecoverable after an audit failure, then the TPM MUST always return TPM_FAILEDSELFTEST for every command other than TPM_GetTestResult. This state must persist regardless of power cycling, the execution of TPM_Init or any other actions.

# 29.6 TPM_GetAuditDigest

***Start of informative comment:***

This returns the current audit digest. The external audit log has the responsibility to track the parameters that constitute the audit digest.

This value may be unique to an individual TPM. The value however will be changing at a rate set by the TPM Owner. Those attempting to use this value may find it changing without their knowledge. This value represents a very poor source of tracking uniqueness.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | | | TPM_COMMAND_CODE | Ordinal | Command ordinal: TPM_ORD_GetAuditDigest |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | Tag | TPM_TAG_RSP_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | | | TPM_RESULT | returnCode | The return code of the operation. |
| 5 | 10 | | | TPM_COUNTER_VALUE | counterValue | The current value of the audit monotonic counter |
| 4 | 20 | | | TPM_DIGEST | auditDigest | Log of all audited events |
| 5 | 4 | | | UINT32 | ordSize | Size of the ordinal list in bytes |
| 6 | <> | | | UINT32[] | ordList | List of ordinals that are audited. |

**Actions**

5. The TPM sets auditDigest to TPM_STANY_DATA -> auditDigest
6. The TPM sets counterValue to TPM_PERMANENT_DATA -> auditMonotonicCounter
7. The TPM sets ordinalList to a list of all audited functions. This is an unordered array of UINT32 values, each entry is the ordinal of an audited command.

# 29.7 TPM_GetAuditDigestSigned

***Start of informative comment:***

The signing of the audit log returns the entire digest value and the list of currently audited commands.

The inclusion of the list of audited commands as an atomic operation is to tie the current digest value with the list of commands that are being audited.

The signing functionality of this command could be handled by a signed transport session. The resetting of the audit log functionality must remain in this command; hence there is no way to remove this ordinal from the set of active ordinals a TPM must support.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | ordinal: TPM_ORD_GetAuditDigestSigned |
| 4 | 4 | | | TPM_KEY_HANDLE | keyHandle | The handle of a loaded key that can perform digital signatures. |
| 5 | 1 | 2s | 1 | BOOL | closeAudit | Indication if audit session should be closed |
| 6 | 20 | 3s | 20 | TPM_NONCE | antiReplay | A nonce to prevent replay attacks |
| 7 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for key authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 8 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 9 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 10 | 20 | | | TPM_AUTHDATA | keyAuth | Authorization HMAC key: key.usageAuth. |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | ordinal: TPM_ORD_GetAuditDigestSigned |
| 4 | 10 | 3S | 10 | TPM_COUNTER_VALUE | counterValue | The value of the audit monotonic counter |
| 5 | 20 | 4S | 20 | TPM_DIGEST | auditDigest | Log of all audited events |
| 6 | 4 | 5S | 4 | UINT32 | ordSize | The size of the ordinal list in bytes |
| 7 | <> | 6S | <> | UINT32[] | ordinalList | The list of ordinals that are being audited |
| 8 | 4 | 7S | 4 | UINT32 | sigSize | The size of the sig parameter |
| 9 | <> | 8S | <> | BYTE[] | sig | The signature of the area |
| 10 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 11 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 12 | 20 | | | TPM_AUTHDATA | resAuth | Authorization. HMAC key: key.usageAuth. |

**Actions**

1. Call TPM_GetAuditDigest and obtain auditDigest, counterValue, ordSize and ordinalList
2. Create D1 a TPM_SIGN_INFO structure and set the structure defaults
   a. Set D1 -> fixed to “ADIG”
   b. Set D1 -> replay to antiReplay
   c. Create D2 the concatenation of auditDigest || counterValue || ordinalList
   d. Set D1 -> dataLen to the length of D2
   e. Set D1 -> data to D2
   f. Create a digital signature of D1 by using the signature scheme for keyHandle
3. If closeAudit == TRUE and keyHandle->keyUsage is TPM_KEY_IDENTITY
   a. TPM_STANY_DATA -> auditDigest MUST be set to NULLS.
4. Else
   a. TPM_INVALID_KEYUSAGE
5. Return the signature in the sig parameter

# 29.8 TPM_SetOrdinalAuditStatus

***Start of informative comment:***

Set the audit flag for a given ordinal. This command requires the authorization of the TPM Owner.

***End of informative comment.***

**Incoming Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RQU_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of input bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_SetOrdinalAuditStatus |
| 4 | 4 | 2S | 4 | TPM_COMMAND_CODE | ordinalToAudit | The ordinal whose audit flag is to be set |
| 5 | 1 | 3S | 1 | BOOL | auditState | Value for audit flag |
| 6 | 4 | | | TPM_AUTHHANDLE | authHandle | The authorization handle used for owner authorization. |
| | | 2H1 | 20 | TPM_NONCE | authLastNonceEven | Even nonce previously generated by TPM to cover inputs |
| 7 | 20 | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 8 | 1 | 4H1 | 1 | BOOL | continueAuthSession | The continue use flag for the authorization handle |
| 9 | 20 | | | TPM_AUTHDATA | ownerAuth | Authorization. HMAC key: ownerAuth. |

**Outgoing Parameters and Sizes**

| PARAM | | HMAC | | Type | Name | Description |
|---|---|---|---|---|---|---|
| # | SZ | # | SZ | | | |
| 1 | 2 | | | TPM_TAG | tag | TPM_TAG_RSP_AUTH1_COMMAND |
| 2 | 4 | | | UINT32 | paramSize | Total number of output bytes including paramSize and tag |
| 3 | 4 | 1S | 4 | TPM_RESULT | returnCode | The return code of the operation. |
| | | 2S | 4 | TPM_COMMAND_CODE | ordinal | Command ordinal: TPM_ORD_SetOrdinalAuditStatus |
| 4 | 20 | 2H1 | 20 | TPM_NONCE | nonceEven | Even nonce newly generated by TPM to cover outputs |
| | | 3H1 | 20 | TPM_NONCE | nonceOdd | Nonce generated by system associated with authHandle |
| 5 | 1 | 4H1 | 1 | BOOL | continueAuthSession | Continue use flag, TRUE if handle is still active |
| 6 | 20 | | | TPM_AUTHDATA | resAuth | Authorization HMAC key: ownerAuth. |

**Actions**

1. Validate the authorization to execute the command and the parameters
2. Validate that the ordinal points to a valid TPM ordinal, return TPM_BAD_INDEX on error
3. Set the non-volatile flag associated with ordinalToAudit to the value in auditState

End of Document – Do Not Delete