TPM Main
Part 3 Commands

Specification Version 1.2
Level 2 Revision 116
1 March 2011
TCG Published

Contact: admin@trustedcomputinggroup.com

TCG Published

TCG

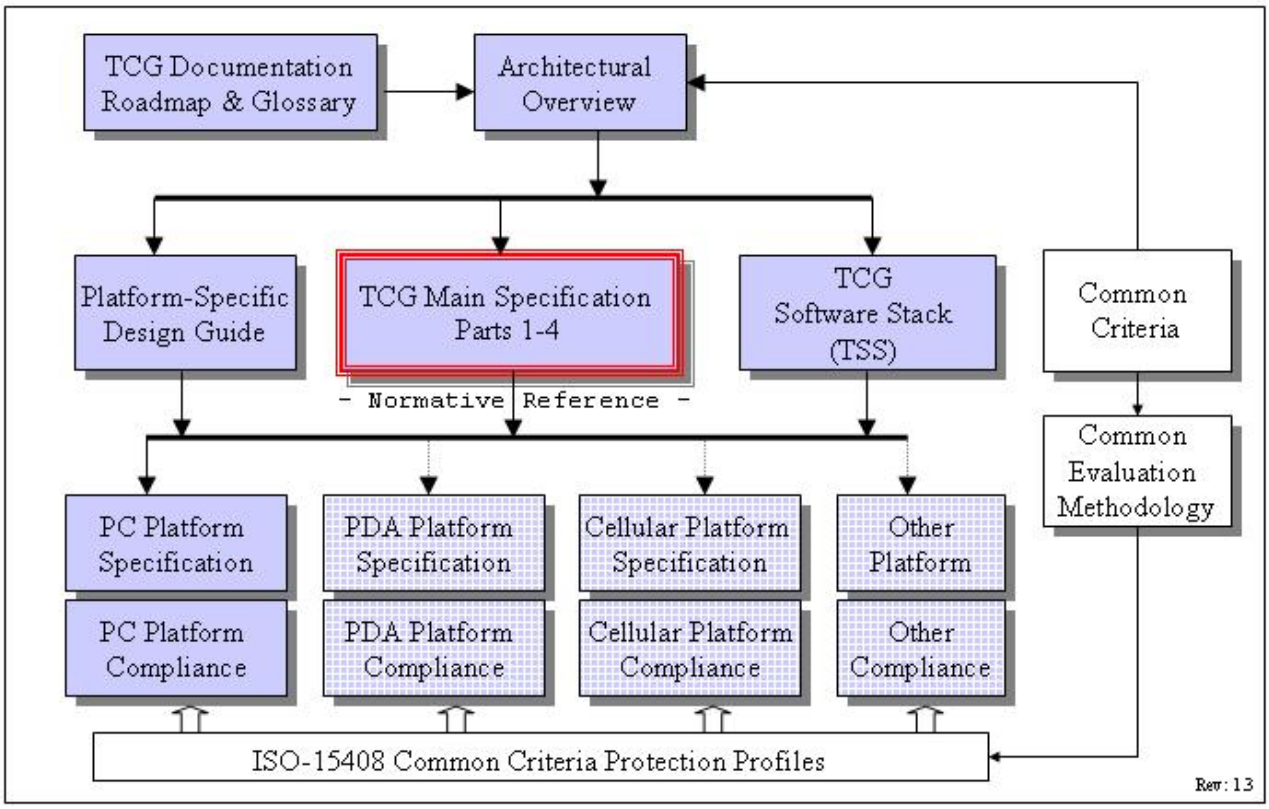
Disclaimers, Notices, and License Terms

58Change History

Version	Date	Description
Rev 50	Jul 2003	Started 01 Jul 2003 by David Grawrock Breakup into parts and the merge of 1.1 commands
Rev 63	Oct 2003	Change history tied to part 1 and kept in part 1 (DP)
Rev 71	Mar 2004	Change in terms from authorization data to AuthData.
Rev 91	Sept 2005	The following modifications were made by Tasneem Brutch: ▪ Update to section 6.2 informative, for TPM_OwnerClear. ▪ Addtion of action item 15, to section 6.2, for TPM_OwnerClear. ▪ Addition of "MAY" to section 20.1, TPM_NV_DefineSpace, Action 1(a). ▪ Addition of a new Action (4) to Section 20.2, TPM_NV_WriteValue ▪ Addtion of a new Action (3) to Section 20.4, TPM_NV_ReadValue. ▪ Typo corrected in Section 21.1 ▪ Moved TPM_GetCapabilityOwner from Section the Deleted Commands (section 28.1) to section 7.3. Added information on operands, command description and actions from Rev. 67.
Rev 92	Sept 2005	Section 7.3 TPM_GetCapabilityOwner Ordinal was added to the outgoing params, which is not returned but is typically included in outParamDigest.
Rev 92	Sept 2005	Corrected a copy and paste error: Part 3 20.2 TPM_NV_WriteValue Removed the Action "3. If D1 -> TPM_NV_PER_AUTHREAD is TRUE return TPM_AUTH_CONFLICT"
Rev 93	Sept. 2005	Moved TPM_CertifySelfTest command to the deleted section.
Rev 100	May 2006	Added deferredPhysicalPresence and its use in TPM_FieldUpgrade, clarified CTR mode, added TPM_NV_INDEX_TRIAL and use in TPM_NV_DefineSpace
Rev 101	Aug 2006	Changed "set to NULL" to "set to all zeros" in many places. TPM_OwnerClear must affect disableFullDALogicInfo. Clarified that _INFO keys may be used where _SHA1 keys are used. Clarified that a global secret can be used for field upgrade confidentiality. Added TPM_CMK_CreateBlob actions for the migrationType parameter. Added TPM_CertifyKey action to check payload. Clarified that TPM_Delegate_LoadOwnerDelegation returns an error if there is no owner and owner authorization is present. Clarified that TPM_NV_DefineSpace cannot define the DIR index. Clarified that the TPM does not have to clean up the effects of a wrapped command upon failure of a transport response. Clarified that TPM_ReleaseCounter does not ignore the continueAuthSession parameter.
Rev 102	Sept 2006	Reworked TPM_GetPubkey to always check authorization data if present and allow no-authorization for TPM_AUTH_PRIV_USE_ONLY or TPM_AUTH_NEVER. Fixed TPM_LoadContext typo, Action 6.e. returns error if the HMAC does NOT match.
Rev 103	Oct 2006	Added warning notes where excluding key handle from HMAC can allow an attack. Added warning that delegating TPM_ChangeAuth allows elevation of privilege.
Rev 104	Nov 2006	Owner clear sets allowMaintenance and readSRKPub to default state. TPM_Unseal can use DSAP. TPM_CreateEndorsementKeyPair uses TPM_ES_RSAESOAEP_SHA1_MGF1.
Rev 105	Feb 2007	TPM_Seal, TPM_CreateWrapKey informative that they lack an identifier. TPM_NV_DefineSpace should check inputs before changing state. TPM_NV_DefineSpace, TPM_NV_WriteValue, TPM_NV_ReadValue ignore disabled and deactivated when nvLocked is FALSE, MAY always check HMAC. TPM_NV_WriteValue must not return error for DIR data size of 0. TPM_NV_ReadValue partial DIR reads are allowed. Informative that audit occurs twice for transport wrapped command. TPM_Reset must invalidate OSAP and DSAP sessions, must not invalidate sessions saved by TPM_SaveContext.
Rev 106	April 2007	Removed tpmProof check for non-migratable parent keys.
Rev 107	July 2007	Removed unused maxNVBufSize. Increment the auditMonotonicCounter before audit response if digest is zero. State should not change on field upgrade authorization failure. PCR values for a key are validated at use, not at load. TPM_StirRandom is not required to check for data < 256 bytes. TPM_ChangeAuth must validate usageAuth. Entity PCRs must be validated each time an OIAP session is used. TPM_ExecuteTransport MUST log public key logs.
Rev 108	Sept 2007	TPM_ForceClear succeeds even with no owner (informative). Audit only occurs when the commands executes successfully. Field upgrade should not change shielded locations. Reordered TPM_NV_DefineSpace, TPM_NV_WriteValue so the NV write cound is not incremented if there is an authorization error.
Rev 109	Oct 2007	Added PCR index check to TPM_SHA1CompleteExtend, TPM_Extend, TPM_PCRRead.
Rev 110	May 2008	Minor typo corrections.

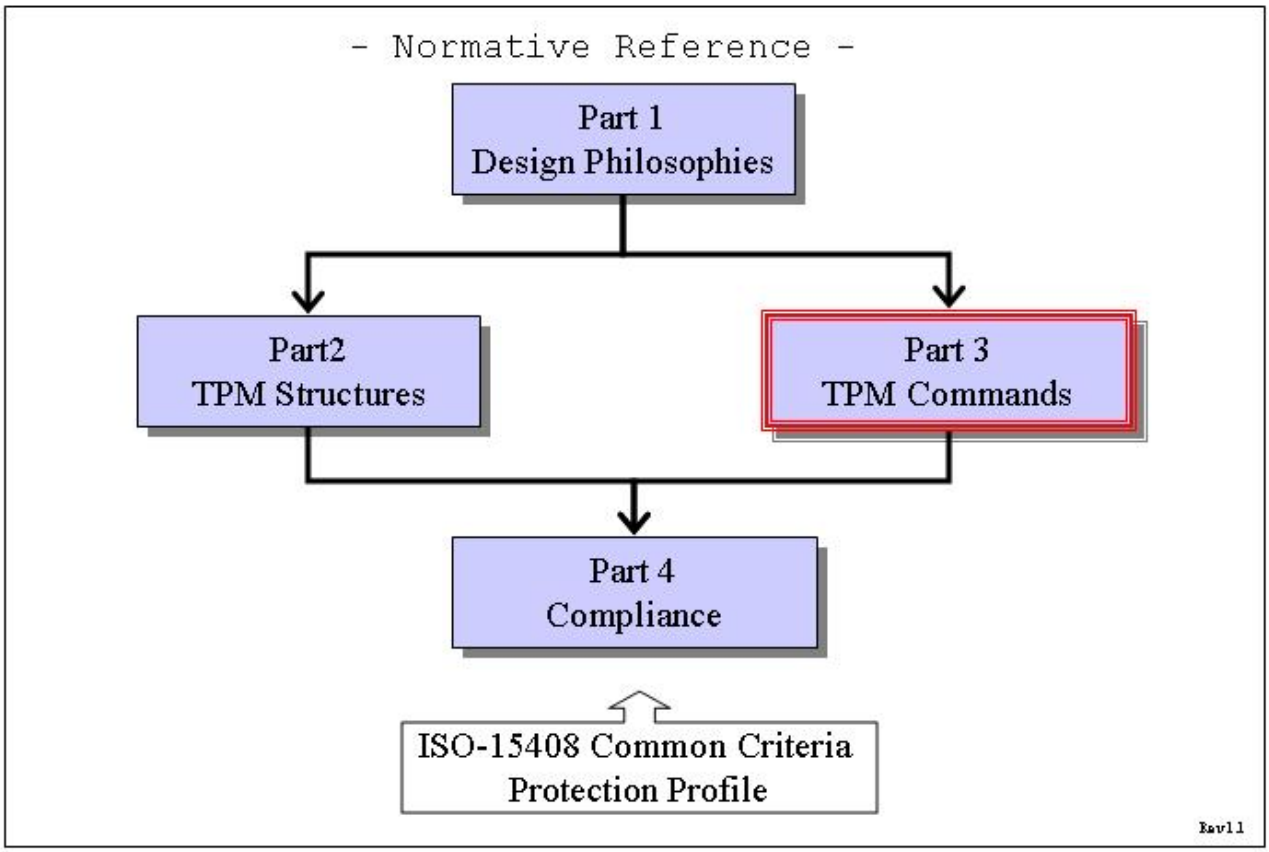
Rev 111	July 2008	TPM_SaveState gives priority to keys where parentPCRStatus is TRUE. Informative security warning when field upgrade adds new features. TPM_MakeIdentity normative that the signing key digestAtRelease is not validated.
Rev 112	Jan 2009	Create and load storage and migrate key, ownership commands check for default exponent, TPM_CMK_CreateBlob MAY check unused restrictTicket and sigTicket, TPM_NV_WriteValue does auth checks before changing bGlobalLock
Rev 113	Jan 2009	Identity key checks for default exponent. TPM_SHA1Update actions added. TPM_NV_WriteValue, TPM_NV_ReadValue added checks for disabled, deactivated.
Rev 114	Jan 2009	No changes
Rev 116	Aug 2009	No Changes

TCG Doc Roadmap – Main Spec



59

TCG Main Spec Roadmap



60

61**Table of Contents**

226

227End of Introduction do not delete

228 1. Scope and Audience

229The TPM main specification is an industry specification that enables trust in computing 230platforms in general. The main specification is broken into parts to make the role of each 231document clear. A version of the specification (like 1.2) requires all parts to be a complete 232specification.

233This is Part 3, the commands that the TPM will use.

234This document is an industry specification that enables trust in computing platforms in 235general.

236**1.1 Key words**

237The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” 238“SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in the chapters 2-10 239normative statements are to be interpreted as described in [RFC-2119].

1.2 Action Order

1. The order of ordinal actions is advisory.

In particular, the order in which errors are checked is vendor specific. The TPM SHOULD check for error conditions as much as possible before executing actions that alter the TPM state.

See also Part 2 “Return Codes” for a discussion of error code requirements.

2. After input parameter parsing errors, the state of the TPM may or may not be changed. The TSS can use TPM_GetCapability to determine what state has been affected.

For example, authorized commands terminate authorization sessions on error, since the response cannot roll the nonce. However, if the incoming session handle parameter cannot be parsed, the session cannot be terminated.

254 1.3 Statement Type

255Please note a very important distinction between different sections of text throughout this 256document. You will encounter two distinctive kinds of text: informative comment and 257normative statements. Because most of the text in this specification will be of the kind 258normative statements, the authors have informally defined it as the default and, as such, 259have specifically called out text of the kind informative comment. They have done this by 260flagging the beginning and end of each informative comment and highlighting its text in 261gray. This means that unless text is specifically marked as of the kind informative 262comment, you can consider it of the kind normative statements.

263For example:

264**Start of informative comment:**

265This is the first paragraph of several paragraphs containing text of the kind informative 266comment …

267This is the second paragraph of text of the kind informative comment …

268This is the nth paragraph of text of the kind informative comment …

269To understand the TPM specification the user must read the specification. (This use of 270MUST does not require any action).

271**End of informative comment.**

272This is the first paragraph of one or more paragraphs (and/or sections) containing the text 273of the kind normative statements …

274To understand the TPM specification the user MUST read the specification. (This use of 275MUST indicates a keyword usage and requires an action).

276 2. Description and TODO

277This document is to show the changes necessary to create the 1.2 version of the TCG 278specification. Some of the sections are brand new text; some are rewritten sections of the 2791.1 version. Upon approval of the 1.2 changes, there will be a merging of the 1.1 and 1.2 280versions 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();
```

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.

3163.2 TPM_Startup

317**Start of informative comment:**

318TPM_Startup is always preceded by TPM_Init, which is the physical indication (a system-
319wide reset) that TPM initialization is necessary.

320There are many events on a platform that can cause a reset and the response to these
321events can require different operations to occur on the TPM. The mere reset indication does
322not contain sufficient information to inform the TPM as to what type of reset is occurring.
323Additional information known by the platform initialization code needs transmitting to the
324TPM. The TPM_Startup command provides the mechanism to transmit the information.

325The TPM can startup in three different modes:

326A “clear” start where all variables go back to their default or non-volatile set state

327A “save” start where the TPM recovers appropriate information and restores various values
328based on a prior TPM_SaveState. This recovery requires an invocation of TPM_Init to be
329successful.

330A failing "save" start must shut down the TPM. The CRTM cannot leave the TPM in a state
331where an untrusted upper software layer could issue a "clear" and then extend PCR's and
332thus mimic the CRTM.

333A “deactivated” start where the TPM turns itself off and requires another TPM_Init before
334the TPM will execute in a fully operational state.

335**End of informative comment.**

336Incoming 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal TPM_ORD_Startup
4	2	2S	2	TPM_STARTUP_TYPE	startupType	Type of startup that is occurring

337Outgoing 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Startup

338Description

339TPM_Startup MUST be generated by a trusted entity (the RTM or the TPM, for example).

340 1. If the TPM is in failure mode

a. TPM_STANY_FLAGS -> postInitialize is still set to FALSE

b. The TPM returns TPM_FAILEDSELFTEST

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, TPM_RT_DAA_TPM, and TPM_RT_TRANS are invalidated
   b. Reset TPM_STCLEAR_DATA -> PCR[] values to each correct default value
      i. pcrReset is FALSE, set to 0x00..00
      ii. pcrReset is TRUE, set to 0xFF..FF
   c. Set the following TPM_STCLEAR_FLAGS to their default state
      i. PhysicalPresence
      ii. PhysicalPresenceLock
      iii. disableForceClear
   d. The TPM MAY initialize auditDigest to all zeros
      i. If not initialized to all zeros, the TPM SHALL ensure that auditDigest contains a valid value.
      ii. If initialization fails, the TPM SHALL set auditDigest to all zeros and SHALL set the internal TPM state so that the TPM returns TPM_FAILEDSELFTEST 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 all zeros
      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 zero
      iii. ownerReference to TPM_KH_OWNER
   h. The TPM MUST set the following TPM_STCLEAR_FLAGS to
      i. bGlobalLock to FALSE
   i. Determine which keys should remain in the TPM
      i. For each key that has a valid preserved value in the TPM

376 (1) if parentPCRStatus is TRUE then call TPM_FlushSpecific(keyHandle)

377 (2) if isVolatile is TRUE then call TPM_FlushSpecific(keyHandle)

378 ii. Keys under control of the OwnerEvict flag MUST stay resident in the TPM

3793. If stType = TPM_ST_STATE

380 a. If the TPM has no state to restore, the TPM MUST set the internal state such that it
381 returns TPM_FAILEDSELFTEST to all subsequent commands.

382 b. The TPM MAY determine for each session type (authorization, transport, DAA, …) to
383 release or maintain the session information. The TPM reports how it manages sessions
384 in the TPM_GetCapability command.

385 c. The TPM SHALL take all necessary actions to ensure that all PCRs contain valid
386 preserved values. If the TPM is unable to successfully complete these actions, it SHALL
387 enter the TPM failure mode.

388 i. For resettable PCR the TPM MUST set the value of TPM_STCLEAR_DATA ->
389 PCR[]to the resettable PCR default value. The TPM MUST NOT restore a resettable
390 PCR to a preserved value

391 d. The TPM MAY initialize auditDigest to all zeros.

392 i. Otherwise, the TPM SHALL take all actions necessary to ensure that
393 auditDigest contains a valid value. If the TPM is unable to successfully complete
394 these actions, the TPM SHALL initialize auditDigest to all zeros and SHALL set the
395 internal state such that the TPM returns TPM_FAILEDSELFTEST to all
396 subsequent commands.

397 e. The TPM MUST restore the following flags to their preserved states:

398 i. All values in TPM_STCLEAR_FLAGS

399 ii. All values in TPM_STCLEAR_DATA

400 f. The TPM MUST restore all keys that have a valid preserved value.

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

4034. If stType = TPM_ST_DEACTIVATED

404 a. Invalidate sessions

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

407 b. Set the TPM_STCLEAR_FLAGS to their default state.

408 c. Set TPM_STCLEAR_FLAGS -> deactivated to TRUE

4095. The TPM MUST ensure that state associated with TPM_SaveState is invalidated

4106. 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.

Resettable PCRs are tied to platform state that does not survive a sleep state. If the PCRs did not reset, they would falsely indicate that the platform state was already present when it came out of sleep. Since some setup is required first, there would be a gap where PCRs indicated the wrong state. Therefore, the PCRs must be recreated.

Any loaded keys may be preserved. Keys with parentPCRStatus TRUE are not given priority because of security concerns. Rather, since the key might be part of a storage tree that requires PCR value transitions, it might not be directly loadable after TPM_Startup(ST_STATE). For a TPM implementation that does not save all loaded keys, the platform should issue a TPM_SaveContext / TPM_LoadContext sequence for those loaded keys. contextNonceKey will be restored, guaranteeing that the saved key context can be restored.

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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveState.

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.

4363. If an explicit action is required to preserve data, it MUST be possible for the TPM to 437 determine whether preserved data is valid.

4384. If a parameter mirrored by any preserved value is altered, all preserved values MUST be 439 declared invalid.

4405. The TPM MAY declare all preserved values invalid in response to any command other 441 than TPM_Init.

442**Actions**

4431. Store TPM_STCLEAR_DATA -> PCR contents except for

444 a. If the PCR attribute pcrReset is TRUE

445 b. Any platform identified debug PCR

4462. The auditDigest MUST be handled according to the audit requirements as reported by 447 TPM_GetCapability.

448 a. If the ordinalAuditStatus is TRUE for the TPM_SaveState ordinal and the auditDigest 449 is being stored in the saved state, the saved auditDigest MUST include the 450 TPM_SaveState input parameters and MUST NOT include the output parameters.

4513. All values in TPM_STCLEAR_DATA MUST be preserved.

4524. All values in TPM_STCLEAR_FLAGS MUST be preserved.

4535. The contents of any key that is currently loaded SHOULD be preserved if the key's 454 parentPCRStatus indicator is TRUE.

4556. The contents of any key that has TPM_KEY_CONTROL_OWNER_EVICT set MUST be 456 preserved

4577. The contents of any key that is currently loaded MAY be preserved.

4588. The contents of sessions (authorization, transport, DAA, etc.) MAY be preserved as 459 reported by TPM_GetCapability.

4. Admin Testing

4.1 TPM_SelfTestFull

Start of informative comment:

TPM_SelfTestFull tests all of the TPM capabilities.

Unlike TPM_ContinueSelfTest, which may optionally return immediately and then perform the tests, TPM_SelfTestFull always performs the tests and then returns success or failure.

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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SelfTestFull

Actions

1. TPM_SelfTestFull SHALL cause a TPM to perform self-test of each TPM internal function.
   a. If the self-test succeeds, return TPM_SUCCESS.
   b. If the self-test fails, return TPM_FAILEDSELFTEST.
2. Failure of any test results in overall failure, and the TPM goes into failure mode.
3. If the TPM has not executed the action of TPM_ContinueSelfTest, the TPM
   a. MAY perform the full self-test.
   b. MAY return TPM_NEEDS_SELFTEST.

4.2 TPM_ContinueSelfTest

Start of informative comment:

TPM_ContinueSelfTest informs the TPM that it should complete the self-test of all TPM functions.

The TPM may return success immediately and then perform the self-test, or it may perform the self-test and then return success or failure.

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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ContinueSelfTest

Description

1. Prior to executing the actions of TPM_ContinueSelfTest, if the TPM receives a command C1 that uses an untested TPM function, the TPM MUST take one of these actions:

   a. The TPM MAY return TPM_NEEDS_SELFTEST

      i. This indicates that the TPM has not tested the internal resources required to execute C1.

      ii. The TPM does not execute C1.

      iii. The caller MUST issue TPM_ContinueSelfTest before re-issuing the command C1.

      (1) If the TPM permits TPM_SelfTestFull prior to completing the actions of TPM_ContinueSelfTest, the caller MAY issue TPM_SelfTestFull rather than TPM_ContinueSelfTest.

   b. The TPM MAY return TPM_DOING_SELFTEST

      i. This indicates that the TPM is doing the actions of TPM_ContinueSelfTest implicitly, as if the TPM_ContinueSelfTest command had been issued.

      ii. The TPM does not execute C1.

iii. The caller MUST wait for the actions of TPM_ContinueSelfTest to complete before reissuing the command C1.

c. The TPM MAY return TPM_SUCCESS or an error code associated with C1.

i. This indicates that the TPM has completed the actions of TPM_ContinueSelfTest and has completed the command C1.

ii. The error code MAY be TPM_FAILEDSELFTEST.

Actions

1. If TPM_PERMANENT_FLAGS -> FIPS is TRUE or TPM_PERMANENT_FLAGS -> TPMpost 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 either

a. MAY immediately return TPM_SUCCESS

i. When TPM_ContinueSelfTest finishes execution, it MUST NOT respond to the caller with a return code.

b. MAY complete the self-test and then return TPM_SUCCESS or TPM_FAILEDSELFTEST.

521 4.3 TPM_GetTestResult

522**Start of informative comment:**

523TPM_GetTestResult provides manufacturer specific information regarding the results of the 524self-test. This command will work when the TPM is in self-test failure mode. The reason for 525allowing this command to operate in the failure mode is to allow TPM manufacturers to 526obtain diagnostic information.

527**End of informative comment.**

528Incoming 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetTestResult

529Outgoing 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetTestResult
4	4	3S	4	UINT32	outDataSize	The size of the outData area
5	<>	4S	<>	BYTE[]	outData	The outData this is manufacturer specific

530Description

531This command will work when the TPM is in self test failure mode or limited operation 532mode.

533Actions

5341. The TPM SHALL respond to this command with a manufacturer specific block of 535 information that describes the result of the latest self-test

5362. 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 PERMANENT 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOwnerInstall
4	1	2S	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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOwnerInstall

Action

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

5505.2 TPM_OwnerSetDisable

551**Start of informative comment:**

552The TPM owner sets the PERMANENT disable flag to TRUE or FALSE.

553**End of informative comment.**

554Incoming 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
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		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 session handle
8	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

555Outgoing 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 session digest for the returned parameters. HMAC key: ownerAuth.

556Action

5571. The TPM SHALL authenticate the command as coming from the TPM Owner. If 558 unsuccessful, the TPM SHALL return TPM_AUTHFAIL.

5592. The TPM SHALL set the TPM_PERMANENT_FLAGS -> disable flag to the value in the 560 disableState parameter.

5613.3 TPM_PhysicalEnable

562**Start of informative comment:**

563Sets the PERMANENT disable flag to FALSE using physical presence as authorization.

564**End of informative comment.**

565Incoming 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalEnable

566Outgoing 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalEnable

567Action

5681. Validate that physical presence is being asserted, if not return TPM_BAD_PRESENCE

5692. The TPM SHALL set the TPM_PERMANENT_FLAGS.disable value to FALSE.

5.4 TPM_PhysicalDisable

Start of informative comment:

Sets the PERMANENT 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	1S	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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalDisable

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.

5795.5 TPM_PhysicalSetDeactivated

580**Start of informative comment:**

581Changes the TPM persistant deactivated flag using physical presence as authorization.

582This command is not available when the TPM is disabled.

583**End of informative comment.**

584Incoming 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalSetDeactivated
4	1	2S	1	BOOL	state	State to which deactivated flag is to be set.

585Outgoing 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PhysicalSetDeactivated

586Action

5871. Validate that physical presence is being asserted, if not return TPM_BAD_PRESENCE

5882. The TPM SHALL set the TPM_PERMANENT_FLAGS.deactivated flag to the value in the 589 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 authentication. The operator can provide the authentication by either the assertion of physical presence or presenting the operator AuthData 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
4	4		4	TPM_AUTHHANDLE	authHandle	Auth handle for operation validation. Session type MUST be OIAP
		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 session handle
7	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_SetTempDeactivated
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 session digest for the returned parameters. HMAC key: operatorAuth.

Action

1. If tag = TPM_TAG_RQU_AUTH1_COMMAND
   a. If TPM_PERMANENT_FLAGS -> operator is FALSE return TPM_NOOPERATOR
   b. 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_BAD_PRESENCE
3. The TPM SHALL set the TPM_STCLEAR_FLAGS.deactivated flag to the value TRUE.

608 5.7 TPM_SetOperatorAuth

609**Start of informative comment:**

610This command allows the setting of the operator AuthData value.

611There is no confidentiality applied to the operator authentication as the value is sent under 612the assumption of being local to the platform. If there is a concern regarding the path 613between the TPM and the keyboard then unless the keyboard is using encryption and a 614secure channel an attacker can read the values.

615**End of informative comment.**

616 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOperatorAuth
4	20	2S	20	TPM_SECRET	operatorAuth	The operator AuthData

617 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SetOperatorAuth

618 Action

6191. If physical presence is not asserted the TPM MUST return TPM_BAD_PRESENCE

6202. The TPM SHALL set the TPM_PERMANENT_DATA -> operatorAuth

6213. The TPM SHALL set TPM_PERMANENT_FLAGS -> operator to TRUE

622 6. Admin Ownership

623 6.1 TPM_TakeOwnership

624 **Start of informative comment:**

625 This command inserts the TPM Ownership value into the TPM.

626 **End of informative comment.**

627 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 AuthData encrypted with PUBEK
7	4	5S	4	UINT32	encSrkAuthSize	The size of the encSrkAuth field
8	<>	6S	<>	BYTE[ ]	encSrkAuth	The SRK AuthData encrypted with PUBEK
9	<>	7S	<>	TPM_KEY	srkParams	Structure containing all parameters of new SRK. pubKey.keyLength & encSize are both 0. This structure MAY be TPM_KEY12.
10	4			TPM_AUTHHANDLE	authHandle	The authorization session 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 session handle
13	20			TPM_AUTHDATA	ownerAuth	Authorization session digest for input params. HMAC key: the new ownerAuth value. See actions for validation operations

628

629Outgoing 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. This structure 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, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: the new ownerAuth value

630Description

631The type of the output srkPub MUST be the same as the type of the input srkParams, either
632both TPM_KEY or both TPM_KEY12.

633Actions

6341. If TPM_PERMANENT_DATA -> ownerAuth is valid return TPM_OWNER_SET

6352. If TPM_PERMANENT_FLAGS -> ownership is FALSE return TPM_INSTALL_DISABLED

6363. If TPM_PERMANENT_DATA -> endorsementKey is invalid return
637 TPM_NO_ENDORSEMENT

6384. Verify that authHandle is of type OIAP on error return TPM_AUTHFAIL

6395. If protocolID is not TPM_PID_OWNER, the TPM MAY return TPM_BAD_PARAMETER

6406. Create A1 a TPM_SECRET by decrypting encOwnerAuth using PRIVEK as the key

641 a. This requires that A1 was encrypted using the PUBEK

642 b. Validate that A1 is a length of 20 bytes, on error return TPM_BAD_KEY_PROPERTY

6437. Validate the command and parameters using A1 and ownerAuth, on error return
644 TPM_AUTHFAIL

6458. Validate srkParams

646 a. If srkParams -> keyUsage is not TPM_KEY_STORAGE return
647 TPM_INVALID_KEYUSAGE

648 b. If srkParams -> migratable is TRUE return TPM_INVALID_KEYUSAGE

649 c. If srkParams -> algorithmParms -> algorithmID is NOT TPM_ALG_RSA return
650 TPM_BAD_KEY_PROPERTY

651 d. If srkParams -> algorithmParms -> encScheme is NOT
652 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. srkParams -> algorithmParms -> parms -> keyLength MUST be greater than or equal to 2048, on error return TPM_BAD_KEY_PROPERTY

g. If srkParams -> algorithmParms -> parms -> exponentSize is not 0, return TPM_BAD_KEY_PROPERTY

h. If TPM_PERMANENT_FLAGS -> FIPS is TRUE

i. If srkParams -> authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS

9. Generate K1 according to the srkParams, on error return TPM_BAD_KEY_PROPERTY

a. This includes copying PCRInfo from srkParams to K1

10. Create A2 a TPM_SECRET by decrypting encSrkAuth using the PRIVEK

a. This requires A2 to be encrypted using the PUBEK

b. Validate that A2 is a length of 20 bytes, on error return TPM_BAD_KEY_PROPERTY

c. Store A2 in K1 -> usageAuth

11. Store K1 in TPM_PERMANENT_DATA -> srk

12. Store A1 in TPM_PERMANENT_DATA -> ownerAuth

13. Create TPM_PERMANENT_DATA -> contextKey according to the rules for the algorithm in use by the TPM to save context blobs

14. Create TPM_PERMANENT_DATA -> delegateKey according to the rules for the algorithm in use by the TPM to save delegate blobs

15. Create TPM_PERMANENT_DATA -> tpmProof by using the TPM RNG

16. Export TPM_PERMANENT_DATA -> srk as srkPub

17. Set TPM_PERMANENT_FLAGS -> readPubek to FALSE

18. Calculate resAuth using the newly established TPM_PERMANENT_DATA -> ownerAuth

6782 6.2 TPM_OwnerClear

679**Start of informative comment:**

680The TPM_OwnerClear command performs the clear operation under Owner authentication.
681This command is available until the Owner executes the TPM_DisableOwnerClear, at which
682time any further invocation of this command returns TPM_CLEAR_DISABLED.

683**End of informative comment.**

684Incoming 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 session handle used for owner authentication.
		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 session digest for inputs and owner authentication. HMAC key: ownerAuth.

685Outgoing 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 session digest for the returned parameters. HMAC key: old ownerAuth.

686Actions

6871. Verify that the TPM Owner authorizes the command and all of the input, on error return
688 TPM_AUTHFAIL.

6892. If TPM_PERMANENT_FLAGS -> disableOwnerClear is TRUE then return
690 TPM_CLEAR_DISABLED.

6913. Unload all loaded keys.

692 a. This includes owner evict keys

693 b. If TPM_PERMANENT_FLAGS -> FIPS is TRUE, the memory locations containing
694 secret or private keys MUST be set to all zeros.

6954. The TPM MUST NOT modify the following TPM_PERMANENT_DATA items

696 a. endorsementKey

697 b. revMajor

698 c. revMinor

699 d. manuMaintPub

700 e. auditMonotonicCounter

701 f. monotonicCounter

702 g. pcrAttrib

703 h. rngState

704 i. EKReset

705 j. lastFamilyID

706 k. tpmDAASeed

707 l. authDIR[0]

708 m. daaProof

709 n. daaBlobKey

7105. The TPM MUST invalidate the following TPM_PERMANENT_DATA items and any internal
711 resources associated with these items

712 a. ownerAuth

713 b. srk

714 c. delegateKey

715 d. delegateTable

716 e. contextKey

717 f. tpmProof

718 g. operatorAuth

7196. The TPM MUST reset to manufacturing defaults the following TPM_PERMANENT_DATA
720 items

721 a. noOwnerNVWrite MUST be set to 0

722 b. ordinalAuditStatus

723 c. restrictDelegate

7247. The TPM MUST invalidate or reset all fields of TPM_STANY_DATA

725 a. Nonces SHALL be reset

726 b. Lists (e.g. contextList) SHALL be invalidated

7278. The TPM MUST invalidate or reset all fields of TPM_STCLEAR_DATA except the PCR's

728 a. Nonces SHALL be reset

729 b. Lists (e.g. contextList) SHALL be invalidated

730 c. deferredPhysicalPresence MUST be set to 0

7319. The TPM MUST set the following TPM_PERMANENT_FLAGS to their default values

732 a. disable

733 b. deactivated

734 c. readPubek

735 d. disableOwnerClear

736 e. disableFullDALogicInfo

737 f. allowMaintenance

738 g. readSRKPub

73910. The TPM MUST set the following TPM_PERMANENT_FLAGS

740 a. ownership to TRUE

741 b. operator to FALSE

742 c. maintenanceDone to FALSE

74311. The TPM releases all TPM_PERMANENT_DATA -> monotonicCounter settings

744 a. This includes invalidating all currently allocated counters. The result will be no
745 currently allocated counters and the new owner will need to allocate counters. The
746 actual count value will continue to increase.

74712. The TPM MUST deallocate all defined NV storage areas where

748 a. TPM_NV_PER_OWNERWRITE is TRUE if nvIndex does not have the “D” bit set

749 b. TPM_NV_PER_OWNERREAD is TRUE if nvIndex does not have the “D” bit set

750 c. The TPM MUST NOT deallocate any other currently defined NV storage areas.

75113. The TPM MUST invalidate all familyTable entries

75214. The TPM MUST terminate all sessions, active or saved.

6.3 TPM_ForceClear

Start of informative comment:

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

TPM_ForceClear can succeed even if no owner is installed. In that case, it does whatever TPM_OwnerClear actions that it can.

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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ForceClear

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 authentication check)

6.4 TPM_DisableOwnerClear

Start of informative comment:

The TPM_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 session handle used for owner authentication.
		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 session handle
7	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. 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 session 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.

7823. When this flag is TRUE the only mechanism that can clear the TPM is the
783 TPM_ForceClear command. The TPM_ForceClear command requires physical access to
784 the TPM to execute.

6.5 TPM_DisableForceClear

Start of informative comment:

The TPM_DisableForceClear command disables the execution of the TPM_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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DisableForceClear

Actions

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

796 6.6 TSC_PhysicalPresence

797 **Start of informative comment:**

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

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

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

809 The command provides two sets of functionality. The first is to enable, permanently, either
810 the HW or the SW ability to assert physical presence. The second is to allow SW, if enabled,
811 to assert physical presence.

812 **End of informative comment.**

813 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TSC_ORD_PhysicalPresence.
4	2	2S	2	TPM_PHYSICAL_PRESENCE	physicalPresence	The state to set the TPM's Physical Presence flags.

814 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TSC_ORD_PhysicalPresence.

815 Actions

816 1. For documentation ease, the bits break into two categories. The first is the lifetime
817 settings and the second is the assertion settings.

818 a. Define A1 to be the lifetime settings: TPM_PHYSICAL_PRESENCE_LIFETIME_LOCK,
819 TPM_PHYSICAL_PRESENCE_HW_ENABLE, TPM_PHYSICAL_PRESENCE_CMD_ENABLE,
820 TPM_PHYSICAL_PRESENCE_HW_DISABLE, and
821 TPM_PHYSICAL_PRESENCE_CMD_DISABLE

b. Define A2 to be the assertion settings: TPM_PHYSICAL_PRESENCE_LOCK, TPM_PHYSICAL_PRESENCE_PRESENT, and TPM_PHYSICAL_PRESENCE_NOTPRESENT

Lifetime lock settings

2. If any A1 setting is present

a. If TPM_PERMANENT_FLAGS -> physicalPresenceLifetimeLock is TRUE, return TPM_BAD_PARAMETER

b. If any A2 setting is present return TPM_BAD_PARAMETER

c. If both physicalPresence -> TPM_PHYSICAL_PRESENCE_HW_ENABLE and physicalPresence -> TPM_PHYSICAL_PRESENCE_HW_DISABLE are TRUE, return TPM_BAD_PARAMETER.

d. If both physicalPresence -> TPM_PHYSICAL_PRESENCE_CMD_ENABLE and physicalPresence -> TPM_PHYSICAL_PRESENCE_CMD_DISABLE are TRUE, return TPM_BAD_PARAMETER.

e. If physicalPresence -> TPM_PHYSICAL_PRESENCE_HW_ENABLE is TRUE Set TPM_PERMANENT_FLAGS -> physicalPresenceHWEnable to TRUE

f. If physicalPresence -> TPM_PHYSICAL_PRESENCE_HW_DISABLE is TRUE Set TPM_PERMANENT_FLAGS -> physicalPresenceHWEnable to FALSE

g. If physicalPresence -> TPM_PHYSICAL_PRESENCE_CMD_ENABLE is TRUE, Set TPM_PERMANENT_FLAGS -> physicalPresenceCMDEnable to TRUE.

h. If physicalPresence -> TPM_PHYSICAL_PRESENCE_CMD_DISABLE is TRUE, Set TPM_PERMANENT_FLAGS -> physicalPresenceCMDEnable to FALSE.

i. If physicalPresence -> TPM_PHYSICAL_PRESENCE_LIFETIME_LOCK is TRUE

    i. Set TPM_PERMANENT_FLAGS -> physicalPresenceLifetimeLock to TRUE

j. Return TPM_SUCCESS

SW physical presence assertion

3. If any A2 setting is present

a. If any A1 setting is present return TPM_BAD_PARAMETER

    i. This check here just for consistency, the prior checks would have already ensured that this was ok

b. If TPM_PERMANENT_FLAGS -> physicalPresenceCMDEnable is FALSE, return TPM_BAD_PARAMETER

c. If both physicalPresence -> TPM_PHYSICAL_PRESENCE_LOCK and physicalPresence -> TPM_PHYSICAL_PRESENCE_PRESENT are TRUE, return TPM_BAD_PARAMETER

d. If both physicalPresence -> TPM_PHYSICAL_PRESENCE_PRESENT and physicalPresence -> TPM_PHYSICAL_PRESENCE_NOTPRESENT are TRUE, return TPM_BAD_PARAMETER

e. If TPM_STCLEAR_FLAGS -> physicalPresenceLock is TRUE, return TPM_BAD_PARAMETER

860 f. If physicalPresence -> TPM_PHYSICAL_PRESENCE_LOCK is TRUE
861 i. Set TPM_STCLEAR_FLAGS -> physicalPresence to FALSE
862 ii. Set TPM_STCLEAR_FLAGS -> physicalPresenceLock to TRUE
863 iii. Return TPM_SUCCESS
864 g. If physicalPresence -> TPM_PHYSICAL_PRESENCE_PRESENT is TRUE
865 i. Set TPM_STCLEAR_FLAGS -> physicalPresence to TRUE
866 h. If physicalPresence -> TPM_PHYSICAL_PRESENCE_NOTPRESENT is TRUE
867 i. Set TPM_STCLEAR_FLAGS -> physicalPresence to FALSE
868 i. Return TPM_SUCCESS
8694. Else // There were no A1 or A2 parameters set
870 a. Return TPM_BAD_PARAMETER

8716.7 TSC_ResetEstablishmentBit

872**Start of informative comment:**

873The PC TPM Interface Specification (TIS) specifies setting tpmEstablished to TRUE upon 874execution of the HASH_START sequence. The setting implies the creation of a Trusted 875Operating System on the platform. Platforms will use the value of tpmEstablished to 876determine if operations necessary to maintain the security perimeter are necessary.

877The tpmEstablished bit provides a non-volatile, secure reporting that a HASH_START was 878previously run on the platform. When a platform makes use of the tpmEstablished bit, the 879platform can reset tpmEstablished as the operation is no longer necessary.

880For example, a platform could use tpmEstablished to ensure that, if HASH_START had ever 881been, executed the platform could use the value to invoke special processing. Once the 882processing is complete the platform will wish to reset tpmEstablished to avoid invoking the 883special process again.

884The TPM_PERMANENT_FLAGS -> tpmEstablished bit described in the TPM specifications 885uses positive logic. The TPM_ACCESS register uses negative logic, so that TRUE is reflected 886as a 0.

887**End of informative comment.**

888Incoming 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TSC_ORD_ResetEstablishmentBit

889Outgoing 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TSC_ORD_ResetEstablishmentBit

890Actions

8911. Validate the assertion of locality 3 or locality 4

8922. Set TPM_PERMANENT_FLAGS -> tpmEstablished to FALSE

8933. Return TPM_SUCCESS

7. The Capability 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 TPM_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 manufacturer and version.

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

The limitation on what can be returned in failure mode restricts the information a manufacturer may return when capArea indicates TPM_CAP_MFR.

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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetCapability
4	4	2S	4	TPM_CAPABILITY_AREA	capArea	Partition of capabilities to be interrogated
5	4	3S	4	UINT32	subCapSize	Size of subCap parameter
6	<>	4S	<>	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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetCapability
4	4	3S	4	UINT32	respSize	The length of the returned capability response
5	<>	4S	<>	BYTE[ ]	resp	The capability response

924**Actions**

9251. The TPM validates the capArea and subCap indicators. If the information is available, 926 the TPM creates the response field and fills in the actual information.

9272. The structure document contains the list of caparea and subCap values

9283. If the TPM is in failure mode or limited operation mode, the TPM MUST return

929 a. TPM_CAP_VERSION

930 b. TPM_CAP_VERSION_VAL

931 c. TPM_CAP_MFR

932 d. TPM_CAP_PROPERTY -> TPM_CAP_PROP_MANUFACTURER

933 e. TPM_CAP_PROPERTY -> TPM_CAP_PROP_DURATION

934 f. TPM_CAP_PROPERTY -> TPM_CAP_PROP_TIS_TIMEOUT

935 g. The TPM MAY return any other capability.

936 7.2 TPM_SetCapability

937**Start of informative comment:**

938This command sets values in the TPM.

939A setValue that is inconsistent with the capArea and subCap is considered a bad 940parameter.

941**End of informative comment.**

942Incoming 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_SetCapability
4	4	2S	4	TPM_CAPABILITY_AREA	capArea	Partition of capabilities to be set
5	4	3S	4	UINT32	subCapSize	Size of subCap parameter
6	<>	4S	<>	BYTE[]	subCap	Further definition of information
7	4	5S	4	UINT32	setValueSize	The size of the value to set
8	<>	6S	<>	BYTE[]	setValue	The value to set
9	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		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 session handle
12	20			TPM_AUTHDATA	ownerAuth	Authorization. HMAC key: owner.usageAuth.

943Outgoing 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_SetCapability
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:owner.usageAuth.

944Actions

9451. If tag = TPM_TAG_RQU_AUTH1_COMMAND, validate the command and parameters 946 using ownerAuth, return TPM_AUTHFAIL on error

9472. The TPM validates the capArea and subCap indicators, including the ability to set value 948 based on any set restrictions

9493. If the capArea and subCap indicators conform with one of the entries in the structure 950 TPM_CAPABILITY_AREA (Values for TPM_SetCapability)

951 a. The TPM sets the relevant flag/data to the value of setValue parameter.

9524. Else

953 a. Return the error code TPM_BAD_PARAMETER.

954 7.3 TPM_GetCapabilityOwner

955**Start of informative comment:**

956TPM_GetCapabilityOwner enables the TPM Owner to retrieve all the non-volatile flags and 957the volatile flags in a single operation.

958The flags summarize many operational aspects of the TPM. The information represented by 959some flags is private to the TPM Owner. So, for simplicity, proof of ownership of the TPM 960must be presented to retrieve the set of flags. When necessary, the flags that are not private 961to the Owner can be deduced by Users via other (more specific) means.

962The normal TPM authentication mechanisms are sufficient to prove the integrity of the 963response. No additional integrity check is required.

964**End of informative comment.**

965 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.

966

967 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	Ordinal: TPM_ORD_GetCapabilityOwner
4	4	3S	4	TPM_VERSION	version	A properly filled out version structure.
5	4	4S	4	UINT32	non_volatile_flags	The current state of the non-volatile flags.
6	4	5S	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.

968

969Description

970For 31>=N>=0

9711. Bit-N of the TPM_PERMANENT_FLAGS structure is the Nth bit after the opening bracket 972 in the definition of TPM_PERMANENT_FLAGS in the version of the specification 973 indicated by the parameter “version”. The bit immediately after the opening bracket is 974 the 0th bit.

9752. Bit-N of the TPM_STCLEAR_FLAGS structure is the Nth bit after the opening bracket in 976 the definition of TPM_STCLEAR_FLAGS in the version of the specification indicated by 977 the parameter “version”. The bit immediately after the opening bracket is the 0th bit.

9783. Bit-N of non_volatile_flags corresponds to the Nth bit in TPM_PERMANENT_FLAGS, and 979 the lsb of non_volatile_flags corresponds to bit0 of TPM_PERMANENT_FLAGS

9804. Bit-N of volatile_flags corresponds to the Nth bit in TPM_STCLEAR_FLAGS, and the lsb 981 of volatile_flags corresponds to bit0 of TPM_STCLEAR_FLAGS

982Actions

9831. The TPM validates that the TPM Owner authorizes the command.

9842. The TPM creates the parameter non_volatile_flags by setting each bit to the same state 985 as the corresponding bit in TPM_PERMANENT_FLAGS. Bits in non_volatile_flags for 986 which there is no corresponding bit in TPM_PERMANENT_FLAGS are set to zero.

9873. The TPM creates the parameter volatile_flags by setting each bit to the same state as the 988 corresponding bit in TPM_STCLEAR_FLAGS. Bits in volatile_flags for which there is no 989 corresponding bit in TPM_STCLEAR_FLAGS are set to zero.

9904. The TPM generates the parameter “version”.

9915. The TPM returns non_volatile_flags, volatile_flags and version to the caller.

8. Auditing

8.1 Audit Generation

Start of informative comment:

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

The TPM maintains an extended value for all audited operations.

End of informative comment.

Description

1. The TPM extends the audit digest whenever the ordinalAuditStatus is TRUE for the ordinal about to be executed.
2. The TPM extends the audit digest only when a command is successfully executed.
   a. If the ordinal is unknown, unimplemented, or cannot be determined, no auditing is performed.
3. Corner cases
   a. TPM_SaveState: Only the input parameters are audited, and the audit occurs before the state is saved. If an error occurs while or after the state is saved, the audit still occurs.
   b. TPM_SetOrdinalAuditStatus: In the case where the ordinalToAudit is TPM_ORD_SetOrdinalAuditStatus, audit is based on the initial state, not the final state.

Actions

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

1. Execute command
   a. Execution implies the performance of the listed actions for the ordinal.
2. If the command will return TPM_SUCCESS
   a. If TPM_STANY_DATA -> auditDigest is all zeros
      i. Increment TPM_PERMANENT_DATA -> auditMonotonicCounter by 1
   b. Create A1 a TPM_AUDIT_EVENT_IN structure
      i. Set A1 -> inputParms to the digest of the input parameters from the command
         1. Digest value according to the HMAC digest rules of the "above the line" parameters (i.e. the first HMAC digest calculation).
      ii. Set A1 -> auditCount to TPM_PERMANENT_DATA -> auditMonotonicCounter

c. Set TPM_STANY_DATA -> auditDigest to SHA-1 (TPM_STANY_DATA -> auditDigest || A1)

d. Create A2 a TPM_AUDIT_EVENT_OUT structure

   i. Set A2 -> outputParms to the digest of the output parameters from the command

      1. Digest value according to the HMAC digest rules of the "above the line" parameters (i.e. the first HMAC digest calculation).

   ii. Set A2 -> auditCount to TPM_PERMANENT_DATA -> auditMonotonicCounter

e. Set TPM_STANY_DATA -> auditDigest to SHA-1 (TPM_STANY_DATA -> auditDigest || A2)

8.2 Effect of audit failing

Start of informative comment:

The TPM audit process could have an internal error when attempting to audit a command. To indicate the audit failure, the TPM will return TPM_AUDITFAIL_SUCCESSFUL. This new functionality changes the 1.1 TPM functionality when this condition occurs.

Since no audit occurs if the command fails, The TPM_AUDITFAIL_UNSUCCESSFUL return code is no longer used.

End of informative comment.

1. When, 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 on subsequent attempts to execute a command.
2. The return code for the command uses the following rules
   a. Command result success, audit success -> return TPM_SUCCESS
   b. Command result failure, no audit -> return command result failure
   c. Command result success, audit failure -> return TPM_AUDITFAIL_SUCCESSFUL
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.

Description

1. This command is never 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

5. Return TPM_STANY_DATA -> auditDigest as auditDigest

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.

Note to future architects

When auditing functionality is active in a TPM, it may seem logical to remove this ordinal from the active set of ordinals as the signing functionality of this command could be handled in a signed transport session. While true, this command has a secondary affect also, resetting the audit log digest. As the reset requires TPM Owner authentication, there must be some way in this command to reflect the TPM Owner wishes. By requiring that a TPM Identity key be the only key that can sign and reset, the TPM Owner's authentication is implicit in the execution of the command (TPM Identity Keys are created and controlled by the TPM Owner only). Hence, while one might want to remove an ordinal this is not one that can be removed if auditing is functional.

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_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 session handle used for key authentication.
		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 session 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	Command 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	20	5S	20	TPM_DIGEST	ordinalDigest	Digest of all audited ordinals
7	4	6S	4	UINT32	sigSize	The size of the sig parameter
8	<>	7S	<>	BYTE[]	sig	The signature of the area
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	Authorization HMAC key: key.usageAuth.

Actions

1. Validate the AuthData and parameters using keyAuth, return TPM_AUTHFAIL on error
2. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING, TPM_KEY_IDENTITY or TPM_KEY_LEGACY, if not return TPM_INVALID_KEYUSAGE
3. The TPM validates that the key pointed to by keyHandle has a signature scheme of TPM_SS_RSASSAPKCS1v15_SHA1 or TPM_SS_RSASSAPKCS1v15_INFO, return TPM_INVALID_KEYUSAGE on error
4. 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 D3 a list of all audited ordinals as defined in the TPM_GetAuditDigest UINT32[] ordList outgoing parameter
   d. Create D4 the SHA-1 of D3
   e. Set auditDigest to TPM_STANY_DATA -> auditDigest
   f. Set counterValue to TPM_PERMANENT_DATA -> auditMonotonicCounter
   g. Create D2 the concatenation of auditDigest || counterValue || D4
   h. Set D1 -> data to D2
   i. Create a digital signature of the SHA-1 of D1 by using the signature scheme for keyHandle
   j. Set ordinalDigest to D4
5. If closeAudit == TRUE

1119 a. If keyHandle->keyUsage is TPM_KEY_IDENTITY

1120 i. TPM_STANY_DATA -> auditDigest MUST be set to all zeros.

1121 b. Else

1122 i. Return TPM_INVALID_KEYUSAGE

8.5 TPM_SetOrdinalAuditStatus

Start of informative comment:

Set the audit flag for a given ordinal. Requires the authentication 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 session handle used for owner authentication.
		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 session handle
9	20			TPM_AUTHDATA	ownerAuth	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 session digest for the returned parameters. HMAC key: ownerAuth.

Actions

1. Validate the AuthData to execute the command and the parameters
2. Validate that the ordinal points to a valid TPM ordinal, return TPM_BADINDEX on error
   a. Valid TPM ordinal means an ordinal that the TPM implementation supports
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.

The policy to determine when it is necessary to perform the actions of TPM_RevokeTrust is outside the TPM spec and determined by other TCG workgroups.

TPM_FieldUpgrade is gated by either owner authorization or deferred assertion of Physical Presence (via the TPM_STCLEAR_DATA -> deferredPhysicalPresence -> unownedFieldUpgrade flag). This gating is acknowledgement that the entity that sets the security policy for a platform must approve field upgrade for that platform. This gating can block a global attack on TPMs when the TPME's privilege information (private key) has been compromised. For blocking to be effective in an unowned TPM, the TPM's ownership flag must be FALSE. (This prevents software from taking ownership and executing TPM_FieldUpgrade with owner authorization.)

If an owner is present, field upgrade MUST be owner authorized, as the actions indicate. This prevents an attacker from using physical presence to upgrade a TPM without detection by the owner.

The advantages of deferred assertion of Physical Presence are that it:

- permits a TPM to be upgraded if taking ownership is undesirable or impractical.
- permits a TPM to be upgraded in the OS environment (where Physical Presence typically cannot be asserted), when the TPM has no owner.

If it is acceptable to take ownership of a TPM temporarily, an alternative to deferred assertion of Physical Presence is the process: (1) take ownership; (2) perform an owner authorized field upgrade; (3) clear the owner from the TPM.

There is no requirement for patch confidentiality. Confidentiality may be implemented using a manufacturer specific mechanism, and may use a global secret such as a symmetric encryption key.

The TPM may set the volatile deactivated flag to TRUE if a reboot is required after the field upgrade. There is no requirement to do so.

The TPM must check owner authorization before changing the TPM state or beginning the upgrade. This prevents a non-owner from mounting a denial-of-service attack. It is understood that a TPM may not be able to stage the entire upgrade patch inside the TPM

before checking owner authorization. That TPM may be forced to move the patch outside the owner authorization HMAC.

Ideally, if the upgrade fails (e.g., due to an authentication failure) the TPM firmware should remain unchanged. It is understood that a TPM may not be able to stage the entire upgrade patch inside the TPM for authentication before beginning the upgrade. On a failure, that TPM may be forced to roll the firmware back to a ROMed version. It may go into an upgrade failure state, where it requires a successful field upgrade before continuing.

If a field upgrade adds or deletes features, the security implications must be analyzed. The associated state must be set to prevent attacks. See, for example, allowMaintenance in Part 2. In addition, if a field upgrade adds maintenance commands, it must atomically install the manufacturer's maintenance public key.

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

Description

The patch integrity and authenticity verification 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 SHOULD only change protected capabilities. The upgrade process SHOULD NOT change shielded locations.

The upgrade process MUST NOT change the disabled or deactivated state from TRUE to FALSE.

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

The execution of a TPM_FieldUpgrade command in a TPM MUST leave the TPM in a state that conforms to a version of TCG's TPM specification and conforms to any extant TCG-defined credentials (certificates) that attest to that upgraded TPM.

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

If the owner authorization fails, the state of the TPM (volatile, nonvolatile, and firmware) MUST remain unchanged. The only exception shall be the dictionary attack mitigation state, which should process the authentication failure.

Actions

The TPM SHALL perform the following when executing the command:

1. If TPM Owner is installed
   a. Validate the command and parameters using TPM owner authentication, return TPM_AUTHFAIL on error
2. Else
   a. If TPM_STCLEAR_DATA -> deferredPhysicalPresence -> unownedFieldUpgrade is FALSE return TPM_BAD_PRESENCE.
3. 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.
4. Validate that the upgrade target is the appropriate TPM model and version.
5. Process the upgrade information and update the protected capabilities
6. 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.
   a. The TPM MAY validate that the upgrade major and minor revision are monotonically increasing.
   b. The TPM MAY allow upgrade with a major and minor revision that is less than currently installed in the TPM.
7. The TPM MAY set the TPM_STCLEAR_FLAGS.deactivated to TRUE

9.2 TPM_SetRedirection

Start of 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 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_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 session 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 session 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	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 session digest for the returned parameters. HMAC key: key.usageAuth

Action

1. If tag == TPM_TAG_RQU_AUTH1_COMMAND

1247 a. Validate the command and parameters using TPM Owner authentication, on error
1248 return TPM_AUTHFAIL

12492. if redirCmd == TPM_REDIR_GPIO

1250 a. Validate that keyHandle points to a loaded key, return TPM_INVALID_KEYHANDLE
1251 on error

1252 b. Validate the key attributes specify redirection, return TPM_BAD_TYPE on error

1253 c. Validate that inputDataSize is 4, return TPM_BAD_PARAM_SIZE on error

1254 d. Validate that inputData points to a valid GPIO channel, return
1255 TPM_BAD_PARAMETER on error

1256 e. Map C1 to the TPM_GPIO_CONFIG_CHANNEL structure indicated by inputData

1257 f. If C1 -> attr specifies TPM_GPIO_ATTR_OWNER

1258 i. If tag != TPM_TAG_RQU_AUTH1_COMMAND return TPM_AUTHFAIL

1259 g. If C1 -> attr specifies TPM_GPIO_ATTR_PP

1260 i. If TPM_STCLEAR_FLAGS -> physicalPresence == FALSE, then return
1261 TPM_BAD_PRESENCE

1262 h. Return TPM_SUCCESS

12633. The TPM MAY support other redirection types. These types may be specified by TCG or
1264 provided by the manufacturer.

9.3 TPM_ResetLockValue

Start of informative comment:

Command that resets the TPM dictionary attack mitigation values

This allows the TPM owner to cancel the effect of a number of successive authorization failures. Dictionary attack mitigation is vendor specific, and the actions here are one possible implementation. The TPM may treat an authorization failure outside the mitigation time as a normal failure and not disable the command.

If this command itself has an authorization failure, it is blocked for the remainder of the lock out period. This prevents a dictionary attack on the owner authorization using this command.

It is understood that this command allows the TPM owner to perform a dictionary attack on other authorization values by alternating a trial and this command. Similarly, delegating this command allows the owner's delegate to perform a dictionary attack.

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_ResetLockValue
4	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for TPM 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 session handle
7	20			TPM_AUTHDATA	ownerAuth	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	Command ordinal: TPM_ORD_ResetLockValue
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	HMAC key: TPM Owner auth

1281**Action**

12821. If TPM_STCLEAR_DATA -> disableResetLock is TRUE return TPM_AUTHFAIL

1283 a. The internal dictionary attack mechanism will set TPM_STCLEAR_DATA ->
1284 disableResetLock to FALSE when the timeout period expires

12852. If the command and parameters validation using ownerAuth fails

1286 a. Set TPM_STCLEAR_DATA -> disableResetLock to TRUE

1287 b. Restart the TPM dictionary attack lock out period

1288 c. Return TPM_AUTHFAIL

12893. Reset the internal TPM dictionary attack mitigation mechanism

1290 a. The mechanism is vendor specific and can include time outs, reboots, and other
1291 mitigation strategies

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.

The sealed data blob does not have a protected identifier. On a platform that does not prevent unauthorized access to data, a data blob can be exchanged by a lower layer without detection. The upper layer software must take additional measures to protect the relation between its identifier of the data blob and the blob itself.

End of informative comment.

1331Incoming 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_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 AuthData for the sealed data.
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 session handle used for keyHandle authorization. Must be an OSAP 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 session digest for inputs and keyHandle. HMAC key: key.usageAuth.

1332Outgoing 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_Seal.
4	<>	3S	<>	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 session digest for the returned parameters. HMAC key: key.usageAuth.

1333Description

1334TPM_Seal is used to encrypt private objects that can only be decrypted using TPM_Unseal.

1335Actions

13361. Validate the authorization to use the key pointed to by keyHandle

13372. If the inDataSize is 0 the TPM returns TPM_BAD_PARAMETER

13383. If the keyUsage field of the key indicated by keyHandle does not have the value
1339 TPM_KEY_STORAGE, the TPM must return the error code TPM_INVALID_KEYUSAGE.

13404. If the keyHandle points to a migratable key then the TPM MUST return the error code
1341 TPM_INVALID_KEY_USAGE.

13425. Determine the version of pcrInfo

1343 a. If pcrInfoSize is 0

1344 i. set V1 to 1

1345 b. Else

1346 i. Point X1 as TPM_PCR_INFO_LONG structure to pcrInfo

1347 ii. If X1 -> tag is TPM_TAG_PCR_INFO_LONG

1348 (1) Set V1 to 2

1349 iii. Else

1350 (1) Set V1 to 1

13516. If V1 is 1 then

1352 a. Create S1 a TPM_STORED_DATA structure

13537. else

1354 a. Create S1 a TPM_STORED_DATA12 structure

1355 b. Set S1 -> et to 0

13568. Set S1 -> encDataSize to 0

13579. Set S1 -> encData to all zeros

135810. Set S1 -> sealInfoSize to pcrInfoSize

135911. If pcrInfoSize is not 0 then

1360 a. if V1 is 1 then

1361 i. Validate pcrInfo as a valid TPM_PCR_INFO structure, return TPM_BADINDEX
1362 on error

1363 ii. Set S1 -> sealInfo -> pcrSelection to pcrInfo -> pcrSelection

1364 iii. Create h1 the composite hash of the PCR selected by pcrInfo -> pcrSelection

1365 iv. Set S1 -> sealInfo -> digestAtCreation to h1

1366 v. Set S1 -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease

1367 b. else

1368 i. Validate pcrInfo as a valid TPM_PCR_INFO_LONG structure, return
1369 TPM_BADINDEX on error

1370 ii. Set S1 -> sealInfo -> creationPCRSelection to pcrInfo -> creationPCRSelection

1371 iii. Set S1 -> sealInfo -> releasePCRSelection to pcrInfo -> releasePCRSelection

1372 iv. Set S1 -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease

v. Set S1 -> sealInfo -> localityAtRelease to pcrInfo -> localityAtRelease

vi. Create h2 the composite hash of the TPM_STCLEAR_DATA -> PCR selected by pcrInfo -> creationPCRSelection

vii. Set S1 -> sealInfo -> digestAtCreation to h2

viii. Set S1 -> sealInfo -> localityAtCreation to TPM_STANY_FLAGS -> localityModifier

12. Create a1 by decrypting encAuth according to the ADIP indicated by authHandle.

13. The TPM provides NO validation of a1. Well-known values (like all zeros) are valid and possible.

14. 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 h3 the SHA-1 of S1

d. Set S2 -> storedDigest to h3

e. Set S2 -> authData to a1

f. Set S2 -> dataSize to inDataSize

g. Set S2 -> data to inData

15. Validate that the size of S2 can be encrypted by the key pointed to by keyHandle, return TPM_BAD_DATASIZE on error

16. Create s3 the encryption of S2 using the key pointed to by keyHandle

17. Set continueAuthSession to FALSE

18. Set S1 -> encDataSize to the size of s3

19. Set S1 -> encData to s3

20. Return S1 as sealedData

10.2 TPM_Unseal

Start of informative comment:

The TPM_Unseal operation will reveal TPM_Seal'ed 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 AuthData 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: 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 session 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 session handle
9	20			TPM_AUTHDATA	parentAuth	The authorization session digest for inputs and parentHandle. HMAC key: parentKey.usageAuth.
10	4			TPM_AUTHHANDLE	dataAuthHandle	The authorization session 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 session digest for the encrypted entity. HMAC key: entity.usageAuth.

1410**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_Unseal.
4	4	3S	4	UINT32	secretSize	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 session 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 session digest used for the dataAuth session. HMAC key: entity.usageAuth.

1411**Actions**

14121. The TPM MUST validate that parentAuth authorizes the use of the key in parentHandle, 1413 on error return TPM_AUTHFAIL

14142. If the keyUsage field of the key indicated by parentHandle does not have the value 1415 TPM_KEY_STORAGE, the TPM MUST return the error code TPM_INVALID_KEYUSAGE.

14163. The TPM MUST check that the TPM_KEY_FLAGS -> Migratable flag has the value FALSE 1417 in the key indicated by parentHandle. If not, the TPM MUST return the error code 1418 TPM_INVALID_KEYUSAGE

14194. Determine the version of inData

1420 a. If inData -> tag = TPM_TAG_STORED_DATA12

1421 i. Set V1 to 2

1422 ii. Map S2 a TPM_STORED_DATA12 structure to inData

1423 b. Else If inData -> ver = 1.1

1424 i. Set V1 to 1

1425 ii. Map S2 a TPM_STORED_DATA structure to inData

1426 c. Else

1427 i. Return TPM_BAD_VERSION

14285. Create d1 by decrypting S2 -> encData using the key pointed to by parentHandle

14296. 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 all zeros

e. Create h1 the SHA-1 of S2

f. d1 -> storedDigest MUST match h1

g. d1 -> payload MUST be TPM_PT_SEAL

h. Any failure MUST return TPM_NOTSEALED_BLOB

7. If S2 -> sealInfoSize 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 TPM_STCLEAR_DATA -> 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. Compare h2 with S2 -> pcrInfo -> digestAtRelease, on mismatch return TPM_WRONGPCRVAL

8. 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.

9. If V1 is 2 and S2 -> et specifies encryption (i.e. is not all zeros) then

a. If tag is not TPM_TAG_RQU_AUTH2_COMMAND, return TPM_AUTHFAIL

b. Verify that the authHandle session type is TPM_PID_OSAP or TPM_PID_DSAP, return TPM_BAD_MODE on error.

c. If the MSB of S2 -> et is TPM_ET_XOR

i. Use MGF1 to create string X1 of length sealedDataSize. The inputs to MGF1 are; authLastnonceEven, nonceOdd, “XOR”, and authHandle -> sharedSecret. The four concatenated values form the Z value that is the seed for MFG1.

ii. Create o1 by XOR of d1 -> data and X1

d. Else

i. Create o1 by encrypting d1 -> data using the algorithm indicated by inData -> et
ii. Key is from authHandle -> sharedSecret
iii. IV is SHA-1 of (authLastNonceEven || nonceOdd)
e. Set continueAuthSession to FALSE

10. else
    a. Set o1 to d1 -> data
11. Set the return secret as o1
12. Return TPM_SUCCESS

10.3 TPM_UnBind

Start of informative comment:

TPM_UnBind takes the data blob that is the result of a Tspi_Data_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.

TPM_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: 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 session handle
10	20			TPM_AUTHDATA	privAuth	The authorization session digest that authorizes the inputs and use of keyHandle. HMAC key: key.usageAuth.

1484Outgoing 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_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 session digest for the returned parameters. HMAC key: key.usageAuth.

1485Description

1486TPM_UnBind SHALL operate on a single block only.

1487Actions

1488The TPM SHALL perform the following:

14891. If the inDataSize is 0 the TPM returns TPM_BAD_PARAMETER

14902. Validate the AuthData to use the key pointed to by keyHandle

14913. If the keyUsage field of the key referenced by keyHandle does not have the value 1492 TPM_KEY_BIND or TPM_KEY_LEGACY, the TPM must return the error code 1493 TPM_INVALID_KEYUSAGE

14944. Decrypt the inData using the key pointed to by keyHandle

14955. if (keyHandle -> encScheme does not equal TPM_ES_RSAESOAEP_SHA1_MGF1) and 1496 (keyHandle -> keyUsage equals TPM_KEY_LEGACY),

1497 a. The payload does not have TPM specific markers to validate, so no consistency check 1498 can be performed.

1499 b. Set the output parameter outData to the value of the decrypted value of inData. 1500 (Padding associated with the encryption wrapping of inData SHALL NOT be returned.)

1501 c. Set the output parameter outDataSize to the size of outData, as deduced from the 1502 decryption process.

15036. else

1504 a. Interpret the decrypted data under the assumption that it is a TPM_BOUND_DATA 1505 structure, and validate that the payload type is TPM_PT_BIND

1506 b. Set the output parameter outData to the value of TPM_BOUND_DATA -> 1507 payloadData. (Other parameters of TPM_BOUND_DATA SHALL NOT be returned. 1508 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.

The key blob does not have a protected identifier. On a platform that does not prevent unauthorized access to data, a key blob can be exchanged by a lower layer without detection. The upper layer software must take additional measures to protect the relation between its identifier of the key blob and the blob itself.

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 AuthData for the key.
6	20	3S	20	TPM_ENCAUTH	dataMigrationAuth	Encrypted migration AuthData for the key.
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	parent key authorization. Must be an OSAP 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	Authorization 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	<>	3S	<>	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	Authorization HMAC key: parentKey.usageAuth.

Actions

The TPM SHALL do the following:

1. Validate the AuthData 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 -> keyFlags -> migratable is TRUE and keyInfo -> keyFlags -> 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 TPM_PERMANENT_FLAGS -> FIPS is TRUE then
   a. If keyInfo -> keySize is less than 1024 return TPM_NOTFIPS
   b. If keyInfo -> authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS
   c. If keyInfo -> keyUsage specifies TPM_KEY_LEGACY return TPM_NOTFIPS
8. If keyInfo -> keyUsage equals TPM_KEY_STORAGE or TPM_KEY_MIGRATE
   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
   v. exponentSize MUST be 0
9. 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
      iii. Validate all remaining TPM_KEY structures
   b. Else if keyInfo -> tag is TPM_TAG_KEY12
      i. Set V1 to 2
      ii. Map wrappedKey to a TPM_KEY12 structure
      iii. Validate all remaining TPM_KEY12 structures

10. Create DU1 by decrypting dataUsageAuth according to the ADIP indicated by authHandle

11. Create DM1 by decrypting dataMigrationAuth according to the ADIP indicated by authHandle

12. Set continueAuthSession to FALSE

13. Generate asymmetric key according to algorithm information in keyInfo

14. 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, the wrappedKey -> encData -> migrationAuth SHALL be set to the value tpmProof

15. If keyInfo->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. Set wrappedKey -> pcrInfo to keyInfo -> pcrInfo

    d. Set wrappedKey -> digestAtCreation to the TPM_COMPOSITE_HASH indicated by creationPCRSelection

    e. If V1 is 2 set wrappedKey -> localityAtCreation to TPM_STANY_DATA -> locality

16. Encrypt the private portions of the wrappedKey structure using the key in parentHandle

17. Return the newly generated key in the wrappedKey parameter

10.5 TPM_LoadKey2

Start of informative comment:

Before the TPM can use a key to either wrap, unwrap, unbind, seal, unseal, sign or perform any other action, it needs to be present in the TPM. The TPM_LoadKey2 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.

To permit this mapping between handle and upper software labels (called key handle virtualization), the key handle returned by TPM_LoadKey2 must not be included in the response HMAC. This may cause problems if several keys are authorized using the same authorization data. Care should be taken to assign different authorization data to each key.

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

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. TPM_Startup with stType == TPM_ST_CLEAR indicates 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.

1619**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_LoadKey2.
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 session 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 session handle
9	20			TPM_AUTHDATA	parentAuth	The authorization session digest for inputs and parentHandle. HMAC key: parentKey.usageAuth.

1620**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_LoadKey2
4	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 session digest for the returned parameters. HMAC key: parentKey.usageAuth.

1621**Actions**

1622The TPM SHALL perform the following steps:

16231. Validate the command and the parameters using parentAuth and parentHandle -> 1624 usageAuth

16252. If parentHandle -> keyUsage is NOT TPM_KEY_STORAGE return 1626 TPM_INVALID_KEYUSAGE

16273. If the TPM is not designed to operate on a key of the type specified by inKey, return the 1628 error code TPM_BAD_KEY_PROPERTY

16294. The TPM MUST handle both TPM_KEY and TPM_KEY12 structures

16305. Decrypt the inKey -> privkey to obtain TPM_STORE_ASYMKEY structure using the key 1631 in parentHandle

26. 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

27. 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 -> migrationAuth equals TPM_PERMANENT_DATA -> tpmProof

e. Validate the mix of encryption and signature schemes

f. If TPM_PERMANENT_FLAGS -> FIPS is TRUE then

i. If keyInfo -> keySize is less than 1024 return TPM_NOTFIPS

ii. If keyInfo -> authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS

iii. If keyInfo -> keyUsage specifies TPM_KEY_LEGACY return TPM_NOTFIPS

g. If inKey -> keyUsage is TPM_KEY_STORAGE or TPM_KEY_MIGRATE

i. algorithmID MUST be TPM_ALG_RSA

ii. Key size MUST be 2048

iii. exponentSize MUST be 0

iv. sigScheme MUST be TPM_SS_NONE

h. If inKey -> keyUsage is TPM_KEY_IDENTITY

i. algorithmID MUST be TPM_ALG_RSA

ii. Key size MUST be 2048

iii. exponentSize MUST be 0

iv. encScheme MUST be TPM_ES_NONE

i. 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.

j. 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

1669 ii. The TPM MUST handle both version 1.1 TPM_PCR_INFO and 1.2
1670 TPM_PCR_INFO_LONG structures according to the type of TPM_KEY structure

1671 (1) The TPM MUST validate the TPM_PCR_INFO or TPM_PCR_INFO_LONG
1672 structures for legal values. However, the digestAtRelease and
1673 localityAtRelease are not validated for authorization until use time.

16748. Perform any processing necessary to make TPM_STORE_ASYMKEY key available for
1675 operations

16769. Load key and key information into internal memory of the TPM. If insufficient memory
1677 exists return error TPM_NOSPACE.

167810. Assign inKeyHandle according to internal TPM rules.

167911. Set InKeyHandle -> parentPCRStatus to parentHandle -> parentPCRStatus.

168012. If parentHandle indicates that it is using PCR registers, then set inKeyHandle ->
1681 parentPCRStatus to TRUE.

1682 10.6 TPM_GetPubKey

1683**Start of informative comment:**

1684The owner of a key may wish to obtain the public key value from a loaded key. This
1685information may have privacy concerns so the command must have authorization from the
1686key owner.

1687**End of informative comment.**

1688Incoming 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_GetPubKey.
4	4			TPM_KEY_HANDLE	keyHandle	TPM handle of key.
5	4			TPM_AUTHHANDLE	authHandle	The authorization session 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 session handle
8	20			TPM_AUTHDATA	keyAuth	Authorization HMAC key: key.usageAuth.

1689Outgoing 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_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.

1690Actions

1691The TPM SHALL perform the following steps:

16921. If tag = TPM_TAG_RQU_AUTH1_COMMAND then

1693 a. Validate the command parameters using keyHandle -> usageAuth, on error return
1694 TPM_AUTHFAIL

16952. Else

1696 a. Verify that keyHandle -> authDataUsage is TPM_NO_READ_PUBKEY_AUTH or
1697 TPM_AUTH_NEVER, on error return TPM_AUTHFAIL

16983. If keyHandle == TPM_KH_SRK then

1699 a. If TPM_PERMANENT_FLAGS -> readSRKPub is FALSE then return
1700 TPM_INVALID_KEYHANDLE

17014. If keyHandle -> pcrInfoSize is not 0

1702 a. If keyHandle -> keyFlags has pcrIgnoredOnRead set to FALSE

1703 i. Create a digestAtRelease according to the specified PCR registers and compare
1704 to keyHandle -> digestAtRelease and if a mismatch return TPM_WRONGPCRVAL

1705 ii. If specified validate any locality requests

17065. Create a TPM_PUBKEY structure and return

10.7 TPM_Sealx

Start of informative comment:

The TPM_Sealx command works exactly like the TPM_Seal command with the additional requirement of encryption for the inData parameter. This command also places in the sealed blob the information that the TPM_Unseal also requires encryption.

TPM_Sealx requires the use of 1.2 data structures. The actions are the same as TPM_Seal without the checks for 1.1 data structure usage.

The method of incrementing the symmetric key counter value is different from that used by some standard crypto libraries (e.g. openSSL, Java JCE) that increment the entire counter value. TPM users should be aware of this to avoid errors when the counter wraps.

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_Sealx
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 AuthData for the sealed data.
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	MUST 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 session handle used for keyHandle authorization. Must be an OSAP 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 session digest for inputs and keyHandle. HMAC key: key.usageAuth.

1719**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_Sealx
4	<>	3S	4	TPM_STORED_DATA	sealedData	Encrypted, integrity-protected data object that is the result of the TPM_Sealx 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 session digest for the returned parameters. HMAC key: key.usageAuth.

1720**Actions**

17211. Validate the authorization to use the key pointed to by keyHandle

17222. If the inDataSize is 0 the TPM returns TPM_BAD_PARAMETER

17233. If the keyUsage field of the key indicated by keyHandle does not have the value 1724 TPM_KEY_STORAGE, the TPM must return the error code TPM_INVALID_KEYUSAGE.

17254. If the keyHandle points to a migratable key then the TPM MUST return the error code 1726 TPM_INVALID_KEY_USAGE.

17275. Create S1 a TPM_STORED_DATA12 structure

17286. Set S1 -> encDataSize to 0

17297. Set S1 -> encData to all zeros

17308. Set S1 -> sealInfoSize to pcrInfoSize

17319. If pcrInfoSize is not 0 then

1732 a. Validate pcrInfo as a valid TPM_PCR_INFO_LONG structure, return TPM_BADINDEX 1733 on error

1734 b. Set S1 -> sealInfo -> creationPCRSelection to pcrInfo -> creationPCRSelection

1735 c. Set S1 -> sealInfo -> releasePCRSelection to pcrInfo -> releasePCRSelection

1736 d. Set S1 -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease

1737 e. Set S1 -> sealInfo -> localityAtRelease to pcrInfo -> localityAtRelease

1738 f. Create h2 the composite hash of the TPM_STCLEAR_DATA -> PCR selected by 1739 pcrInfo -> creationPCRSelection

1740 g. Set S1 -> sealInfo -> digestAtCreation to h2

1741 h. Set S1 -> sealInfo -> localityAtCreation to TPM_STANY_DATA -> localityModifier

174210. Create S2 a TPM_SEALED_DATA structure

11. Create a1 by decrypting encAuth according to the ADIP indicated by authHandle.
    a. If authHandle indicates XOR encryption for the AuthData secrets
        i. Set S1 -> et to TPM_ET_XOR || TPM_ET_KEY
        (1) TPM_ET_KEY is added because TPM_Unseal uses zero as a special value indicating no encryption.
    b. Else
        i. Set S1 -> et to the algorithm indicated by authHandle
12. The TPM provides NO validation of a1. Well-known values (like all zeros) are valid and possible.
13. If authHandle indicates XOR encryption
    a. Use MGF1 to create string X2 of length inDataSize. The inputs to MGF1 are; authLastNonceEven, nonceOdd, "XOR", and authHandle -> sharedSecret. The four concatenated values form the Z value that is the seed for MFG1.
    b. Create o1 by XOR of inData and X2
14. Else
    a. Create o1 by decrypting inData using the algorithm indicated by authHandle
    b. Key is from authHandle -> sharedSecret
    c. CTR is SHA-1 of (authLastNonceEven || nonceOdd)
15. 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 h3 the SHA-1 of S1
    d. Set S2 -> storedDigest to h3
    e. Set S2 -> authData to a1
    f. Set S2 -> dataSize to inDataSize
    g. Set S2 -> data to o1
16. Validate that the size of S2 can be encrypted by the key pointed to by keyHandle, return TPM_BAD_DATASIZE on error
17. Create s3 the encryption of S2 using the key pointed to by keyHandle
18. Set continueAuthSession to FALSE
19. Set S1 -> encDataSize to the size of s3
20. Set S1 -> encData to s3
21. Return S1 as sealedData

11. Migration

Start of informative comment:

The migration of a key from one TPM to another is a vital aspect to many use models of the TPM. The migration commands are the commands that allow this operation to occur.

There are two types of migratable keys, the version 1.1 migratable keys and the version 1.2 certifiable migratable keys.

End of informative comment.

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 directly to move the key to a new parent (on either 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 AuthData 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 session 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 session 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 session 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.

1804Outgoing 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	continueEntitySession	Continue use flag for entity session
13	20			TPM_AUTHDATA	entityAuth	Authorization HMAC key: entity.migrationAuth.

1805Description

1806The TPM does not check the PCR values when migrating values locked to a PCR.

1807The second authorization session (using entityAuth) MUST be OIAP because OSAP does not
1808have a suitable entityType

1809Actions

18101. Validate that parentAuth authorizes the use of the key pointed to by parentHandle.

18112. Validate that parentHandle -> keyUsage is TPM_KEY_STORAGE, if not return
1812 TPM_INVALID_KEYUSAGE

18133. Create d1 a TPM_STORE_ASYMKEY structure by decrypting encData using the key
1814 pointed to by parentHandle.

1815 a. Verify that d1 -> payload is TPM_PT_ASYM.

18164. Validate that entityAuth authorizes the migration of d1. The validation MUST use d1 ->
1817 migrationAuth as the secret.

18185. Validate that migrationKeyAuth -> digest is the SHA-1 hash of (migrationKeyAuth ->
1819 migrationKey || migrationKeyAuth -> migrationScheme || TPM_PERMANENT_DATA ->
1820 tpmProof).

18216. If migrationType == TPM_MS_MIGRATE the TPM SHALL perform the following actions:

1822 a. Build two byte arrays, K1 and K2:

1823 i. K1 = d1.privKey[0..19] (d1.privKey.keyLength + 16 bytes of d1.privKey.key), sizeof(K1) = 20

1825 ii. K2 = d1.privKey[20..131] (position 16-127 of d1 . privKey.key), sizeof(K2) = 112

1826 b. Build M1 a TPM_MIGRATE_ASYMKEY structure

1827 i. TPM_MIGRATE_ASYMKEY.payload = TPM_PT_MIGRATE

1828 ii. TPM_MIGRATE_ASYMKEY.usageAuth = d1.usageAuth

1829 iii. TPM_MIGRATE_ASYMKEY.pubDataDigest = d1. pubDataDigest

1830 iv. TPM_MIGRATE_ASYMKEY.partPrivKeyLen = 112 – 127.

1831 v. TPM_MIGRATE_ASYMKEY.partPrivKey = K2

1832 c. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the
1833 OAEP encoding of m using OAEP parameters of

1834 i. m = M1 the TPM_MIGRATE_ASYMKEY structure

1835 ii. pHash = d1->migrationAuth

1836 iii. seed = s1 = K1

1837 d. Create r1 a random value from the TPM RNG. The size of r1 MUST be the size of o1.
1838 Return r1 in the Random parameter.

1839 e. Create x1 by XOR of o1 with r1

1840 f. Copy r1 into the output field “random”.

1841 g. Encrypt x1 with the migration public key included in migrationKeyAuth.

18427. If migrationType == TPM_MS_REWRAP the TPM SHALL perform the following actions:

1843 a. Rewrap the key using the public key in migrationKeyAuth, keeping the existing
1844 contents of that key.

1845 b. Set randomSize to 0 in the output parameter array

18468. Else

1847 a. Return TPM_BAD_PARAMETER

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 TPM_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 session 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 session handle
12	20			TPM_AUTHDATA	parentAuth	The authorization session digest that authorizes the inputs and the migration of the key in parentHandle. HMAC key: parentKey.usageAuth

1860**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 session digest for the returned parameters. HMAC key: parentKey.usageAuth

1861**Action**

1862The TPM SHALL perform the following:

18631. Validate the AuthData to use the key in parentHandle

18642. If the keyUsage field of the key referenced by parentHandle does not have the value 1865 TPM_KEY_STORAGE, the TPM must return the error code TPM_INVALID_KEYUSAGE

18663. Create d1 by decrypting the inData area using the key in parentHandle

18674. Create o1 by XOR d1 and random parameter

18685. Create m1 a TPM_MIGRATE_ASYMKEY structure, seed and pHash by OAEP decoding o1

18696. Create k1 by combining seed and the TPM_MIGRATE_ASYMKEY -> partPrivKey field

18707. Create d2 a TPM_STORE_ASYMKEY structure

1871 a. Verify that m1 -> payload == TPM_PT_MIGRATE

1872 b. Set d2 -> payload = TPM_PT_ASYM

1873 c. Set d2 -> usageAuth to m1 -> usageAuth

1874 d. Set d2 -> migrationAuth to pHash

1875 e. Set d2 -> pubDataDigest to m1 -> pubDataDigest

1876 f. Set d2 -> privKey field to k1

18778. Create outData using the key in parentHandle to perform the encryption

1878 11.3 TPM_AuthorizeMigrationKey

1879**Start of informative comment:**

1880This command creates an authorization blob, to allow the TPM owner to specify which 1881migration facility they will use and allow users to migrate information without further 1882involvement with the TPM owner.

1883It is the responsibility of the TPM Owner to determine whether migrationKey is appropriate 1884for migration. The TPM checks just the cryptographic strength of migrationKey.

1885**End of informative comment.**

1886 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_AuthorizeMigrationKey
4	2	2S	2	TPM_MIGRATE_SCHEME	migrationScheme	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 session handle used for owner authentication.
		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 session handle
8	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authorization. HMAC key: ownerAuth.

1887 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_AuthorizeMigrationKey
4	<>	3S	<>	TPM_MIGRATIONKEYAUTH	outData	Returned public key and authorization session 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 session digest for the returned parameters. HMAC key: ownerAuth.

1888

1889 Action

1890The TPM SHALL perform the following:

18911. Check that the cryptographic strength of migrationKey is at least that of a 2048 bit RSA
1892 key. If migrationKey is an RSA key, this means that migrationKey MUST be 2048 bits or
1893 greater and MUST use the default exponent.

18942. Validate the AuthData to use the TPM by the TPM Owner

18953. Create a f1 a TPM_MIGRATIONKEYAUTH structure

18964. Verify that migrationKey-> algorithmParms -> encScheme is
1897 TPM_ES_RSAESOAEP_SHA1_MGF1, and return the error code
1898 TPM_INAPPROPRIATE_ENC if it is not

18995. Set f1 -> migrationKey to the input migrationKey

19006. Set f1 -> migrationScheme to the input migrationScheme

19017. Create v1 by concatenating (migrationKey || migrationScheme ||
1902 TPM_PERMANENT_DATA -> tpmProof)

19038. Create h1 by performing a SHA-1 hash of v1

19049. Set f1 -> digest to h1

190510. Return f1 as outData

11.4 TPM_MigrateKey

Start of informative comment:

The TPM_MigrateKey command performs the function of a migration authority.

The command is relatively simple; it just decrypts the input packet (coming from TPM_CreateMigrationBlob or TPM_CMK_CreateBlob) and then re-encrypts it with the input public key. The output of this command would then be sent to TPM_ConvertMigrationBlob or TPM_CMK_ConvertMigration on the target TPM.

TPM_MigrateKey does not make ANY assumptions about the contents of the encrypted blob. Since it does not have the XOR string, it cannot actually determine much about the key that is being migrated.

This command exists to permit the TPM to be a migration authority. If used in this way, it is expected that the physical security of the system containing the TPM and the AuthData value for the MA key would be tightly controlled.

To prevent the execution of this command using any other key as a parent key, this command works only if keyUsage for maKeyHandle is TPM_KEY_MIGRATE.

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_MigrateKey
4	4			TPM_KEY_HANDLE	maKeyHandle	Handle of the key to be used to migrate the key.
5	<>	2S	<>	TPM_PUBKEY	pubKey	Public key to which the blob is to be migrated
6	4	3S	4	UINT32	inDataSize	The size of inData
7	<>	4S	<>	BYTE[ ]	inData	The input blob
8	4			TPM_AUTHHANDLE	maAuthHandle	The authorization session handle used for maKeyHandle.
		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 session handle
11	20			TPM_AUTHDATA	keyAuth	The authorization session digest for the inputs and key to be signed. HMAC key: maKeyHandle.usageAuth.

1923**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_MigrateKey
4	4	3S	4	UINT32	outDataSize	The used size of the output area for outData
5	<>	4S	<>	BYTE[ ]	outData	The re-encrypted blob
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with certAuthHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag for cert key session
8	20			TPM_AUTHDATA	keyAuth	The authorization session digest for the target key. HMAC key: maKeyHandle.usageAuth

1924**Actions**

19251. Validate that keyAuth authorizes the use of the key pointed to by maKeyHandle

19262. The TPM validates that the key pointed to by maKeyHandle has a key usage value of 1927 TPM_KEY_MIGRATE, and that the allowed encryption scheme is 1928 TPM_ES_RSAESOAEP_SHA1_MGF1.

19293. The TPM validates that pubKey is of a size supported by the TPM and that its size is 1930 consistent with the input blob and maKeyHandle.

19314. The TPM decrypts inData and re-encrypts it using pubKey.

11.5 TPM_CMK_SetRestrictions

Start of informative comment:

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

This command is provided for privacy reasons and must not itself be delegated, because a certified-migration-key may involve a contractual relationship between the Owner and an external entity.

Since restrictions are validated at DSAP session use, there is no need to invalidate DSAP sessions when the restriction value changes.

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_DELEGATE	restriction	The bit mask of how to set the restrictions on CMK keys
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle TPM Owner authentication
		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 session handle
8	20			TPM_AUTHDATA	ownerAuth	The authorization session digest. 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
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.

Description

TPM_PERMANENT_DATA -> restrictDelegate is used as follows

1. If the session type is TPM_PID_DSAP and TPM_KEY -> keyFlags -> migrateAuthority is TRUE

a. If

TPM_KEY_USAGE is TPM_KEY_SIGNING and restrictDelegate -> TPM_CMK_DELEGATE_SIGNING is TRUE, or

TPM_KEY_USAGE is TPM_KEY_STORAGE and restrictDelegate -> TPM_CMK_DELEGATE_STORAGE is TRUE, or

TPM_KEY_USAGE is TPM_KEY_BIND and restrictDelegate -> TPM_CMK_DELEGATE_BIND is TRUE, or

TPM_KEY_USAGE is TPM_KEY_LEGACY and restrictDelegate -> TPM_CMK_DELEGATE_LEGACY is TRUE, or

TPM_KEY_USAGE is TPM_KEY_MIGRATE and restrictDelegate -> TPM_CMK_DELEGATE_MIGRATE is TRUE

then the key can be used.

b. Else return TPM_INVALID_KEYUSAGE.

Actions

1. Validate the ordinal and parameters using TPM Owner authentication, return TPM_AUTHFAIL on error
2. Set TPM_PERMANENT_DATA -> TPM_CMK_DELEGATE -> restrictDelegate = restriction
3. Return TPM_SUCCESS

1966 11.6 TPM_CMK_ApproveMA

1967**Start of informative comment:**

1968This command creates an authorization ticket, to allow the TPM owner to specify which 1969Migration Authorities they approve and allow users to create certified-migration-keys 1970without further involvement with the TPM owner.

1971It is the responsibility of the TPM Owner to determine whether a particular Migration 1972Authority is suitable to control migration

1973**End of informative comment.**

1974 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_ApproveMA
4	20	2S	20	TPM_DIGEST	migrationAuthorityDigest	A digest of a TPM_MSA_COMPOSITE structure (itself one or more digests of public keys belonging to migration authorities)
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		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 session handle
8	20			TPM_AUTHDATA	ownerAuth	Authorization HMAC, key: ownerAuth.

1975 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_ApproveMA
4	20	3S	20	TPM_HMAC	outData	HMAC of migrationAuthorityDigest
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.

1976 Action

1977The TPM SHALL perform the following:

19781. Validate the AuthData to use the TPM by the TPM Owner

19792. Create M2 a TPM_CMK_MA_APPROVAL structure

a. Set M2 ->migrationAuthorityDigest to migrationAuthorityDigest

13. Set outData = HMAC(M2) using tpmProof as the secret

14. Return TPM_SUCCESS

1983 11.7 TPM_CMK_CreateKey

1984**Start of informative comment:**

1985The TPM_CMK_CreateKey command both generates and creates a secure storage bundle for
1986asymmetric keys whose migration is controlled by a migration authority.

1987TPM_CMK_CreateKey is very similar to TPM_CreateWrapKey, but: (1) the resultant key must
1988be a migratable key and can be migrated only by TPM_CMK_CreateBlob; (2) the command is
1989Owner authorized via a ticket.

1990TPM_CMK_CreateKey creates an otherwise normal migratable key except that (1)
1991migrationAuth is an HMAC of the migration authority and the new key's public key, signed
1992by tpmProof (instead of being tpmProof); (2) the migrationAuthority bit is set TRUE; (3) the
1993payload type is TPM_PT_MIGRATE_RESTRICTED.

1994The migration-selection/migration authority is specified by passing in a public key (actually
1995the digests of one or more public keys, so more than one migration authority can be
1996specified).

1997**End of informative comment.**

1998 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_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 AuthData for the key.
6	<>	3S	<>	TPM_KEY12	keyInfo	Information about key to be created, pubkey.keyLength and keyInfo.encData elements are 0. MUST be TPM_KEY12
7	20	4S	20	TPM_HMAC	migrationAuthorityApproval	A ticket, created by the TPM Owner using TPM_CMK_ApproveMA, approving a TPM_MSA_COMPOSITE structure
8	20	5S	20	TPM_DIGEST	migrationAuthorityDigest	The digest of a TPM_MSA_COMPOSITE structure
9	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for parent key authorization. Must be an OSAP session.
		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	Ignored
12	20			TPM_AUTHDATA	pubAuth	The authorization session digest that authorizes the use of the public key in parentHandle. HMAC key: parentKey.usageAuth.

1999**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_CreateKey
4	<>	3S	<>	TPM_KEY12	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 session digest for the returned parameters. HMAC key: parentKey.usageAuth.

2000**Actions**

2001The TPM SHALL do the following:

20021. Validate the AuthData to use the key pointed to by parentHandle. Return
2003 TPM_AUTHFAIL on any error

20042. Validate the session type for parentHandle is OSAP

20053. If the TPM is not designed to create a key of the type requested in keyInfo, return the
2006 error code TPM_BAD_KEY_PROPERTY

20074. Verify that parentHandle->keyUsage equals TPM_KEY_STORAGE

20085. Verify that parentHandle-> keyFlags-> migratable == FALSE

20096. If keyInfo -> keyFlags -> migratable is FALSE, return TPM_INVALID_KEYUSAGE

20107. If keyInfo -> keyFlags -> migrateAuthority is FALSE , return TPM_INVALID_KEYUSAGE

20118. Verify that the migration authority is authorized

2012 a. Create M1 a TPM_CMK_MA_APPROVAL structure

2013 i. Set M1 ->migrationAuthorityDigest to migrationAuthorityDigest

2014 b. Verify that migrationAuthorityApproval == HMAC(M1) using tpmProof as the secret
2015 and return error TPM_MA_AUTHORITY on mismatch

20169. Validate key parameters

2017 a. keyInfo -> keyUsage MUST NOT be TPM_KEY_IDENTITY or
2018 TPM_KEY_AUTHCHANGE. If it is, return TPM_INVALID_KEYUSAGE

201910. If TPM_PERMANENT_FLAGS -> FIPS is TRUE then

2020 a. If keyInfo -> keySize is less than 1024 return TPM_NOTFIPS

2021 b. If keyInfo -> authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS

2022 c. If keyInfo -> keyUsage specifies TPM_KEY_LEGACY return TPM_NOTFIPS

202311. If keyInfo -> keyUsage equals TPM_KEY_STORAGE or TPM_KEY_MIGRATE

2024 a. algorithmID MUST be TPM_ALG_RSA

2025 b. encScheme MUST be TPM_ES_RSAESOAEP_SHA1_MGF1

2026 c. sigScheme MUST be TPM_SS_NONE

2027 d. key size MUST be 2048

2028 e. exponentSize MUST be 0

202912. If keyInfo -> tag is NOT TPM_TAG_KEY12 return error TPM_INVALID_STRUCTURE

203013. Map wrappedKey to a TPM_KEY12 structure

203114. Create DU1 by decrypting dataUsageAuth according to the ADIP indicated by 2032 authHandle.

203315. Set continueAuthSession to FALSE

203416. Generate asymmetric key according to algorithm information in keyInfo

203517. Fill in the wrappedKey structure with information from the newly generated key.

2036 a. Set wrappedKey -> encData -> usageAuth to DU1

2037 b. Set wrappedKey -> encData -> payload to TPM_PT_MIGRATE_RESTRICTED

2038 c. Create thisPubKey, a TPM_PUBKEY structure containing wrappedKey's public key 2039 and algorithm parameters

2040 d. Create M2 a TPM_CMK_MIGAUTH structure

2041 i. Set M2 -> msaDigest to migrationAuthorityDigest

2042 ii. Set M2 -> pubKeyDigest to SHA-1 (thisPubKey)

2043 e. Set wrappedKey -> encData -> migrationAuth equal to HMAC(M2), using tpmProof as 2044 the shared secret

204518. If keyInfo->PCRInfoSize is non-zero

2046 a. Set wrappedKey -> pcrInfo to a TPM_PCR_INFO_LONG structure

2047 b. Set wrappedKey -> pcrInfo to keyInfo -> pcrInfo

2048 c. Set wrappedKey -> digestAtCreation to the TPM_COMPOSITE_HASH indicated by 2049 creationPCRSelection

2050 d. Set wrappedKey -> localityAtCreation to TPM_STANY_FLAGS -> localityModifier

205119. Encrypt the private portions of the wrappedKey structure using the key in parentHandle

205220. Return the newly generated key in the wrappedKey parameter

11.8 TPM_CMK_CreateTicket

Start of informative comment:

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

TPM_CMK_CreateTicket 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	<>	2S	<>	TPM_PUBKEY	verificationKey	The public key to be used to check signatureValue
5	20	3S	20	TPM_DIGEST	signedData	The data to be verified
6	4	4S	4	UINT32	signatureValueSize	The size of the signatureValue
7	<>	5S	<>	BYTE[]	signatureValue	The signatureValue to be verified
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		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 session digest for inputs and owner. HMAC key: ownerAuth.

2061**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_HMAC	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.

2062**Actions**

2063The TPM SHALL do the following:

20641. Validate the TPM Owner authentication to use the command

20652. Validate that the key type and algorithm are correct

2066 a. Validate that verificationKey -> algorithmParms -> algorithmID == TPM_ALG_RSA

2067 b. Validate that verificationKey -> algorithmParms ->encScheme == TPM_ES_NONE

2068 c. Validate that verificationKey ->algorithmParms ->sigScheme is
2069 TPM_SS_RSASSAPKCS1v15_SHA1 or TPM_SS_RSASSAPKCS1v15_INFO

20703. Use verificationKey to verify that signatureValue is a valid signature on signedData, and
2071 return error TPM_BAD_SIGNATURE on mismatch

20724. Create M2 a TPM_CMK_SIGTICKET

2073 a. Set M2 -> verKeyDigest to the SHA-1 (verificationKey)

2074 b. Set M2 -> signedData to signedData

20755. Set sigTicket = HMAC(M2) signed by using tpmProof as the secret

20766. Return TPM_SUCCESS

2077 11.9 TPM_CMK_CreateBlob

2078**Start of informative comment:**

2079TPM_CMK_CreateBlob command is very similar to TPM_CreateMigrationBlob, except that it:
2080(1) uses an extra ticket (restrictedKeyAuth) instead of a migrationAuth authorization
2081session; (2) uses the migration options TPM_MS_RESTRICT_MIGRATE or
2082TPM_MS_RESTRICT_APPROVE; (3) produces a wrapped key blob whose migrationAuth is
2083independent of tpmProof.

2084If the destination (parent) public key is the MA, migration is implicitly permitted. Further
2085checks are required if the MA is not the destination (parent) public key, and merely selects
2086a migration destination: (1) sigTicket must prove that restrictTicket was signed by the MA;
2087(2) restrictTicket must vouch that the target public key is approved for migration to the
2088destination (parent) public key. (Obviously, this more complex method may also be used by
2089an MA to approve migration to that MA.) In both cases, the MA must be one of the MAs
2090implicitly listed in the migrationAuth of the target key-to-be-migrated.

2091When the migrationType is TPM_MS_RESTRICT_MIGRATE, restrictTicket and sigTicket are
2092unused. The TPM may test that the corresponding sizes are zero, so the caller should set
2093them to zero for interoperability.

2094**End of informative comment.**

2095Incoming 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 session 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	msaListSize	The size of the msaList parameter, which is a variable length TPM_MSA_COMPOSITE structure
9	<>	6S	<>	TPM_MSA_COMPOSITE	msaList	One or more digests of public keys belonging to migration authorities
10	4	7S	4	UINT32	restrictTicketSize	The size of the restrictTicket parameter
11	<>	8S	<>	BYTE[]	restrictTicket	If migrationType is TPM_MS_RESTRICT_APPROVE, a TPM_CMK_AUTH structure, containing the digests of the public keys belonging to the Migration Authority, the destination parent key and the key-to-be-migrated.
12	4	9S	4	UINT32	sigTicketSize	The size of the sigTicket parameter
13	<>	10S	<>	BYTE[]	sigTicket	If migrationType is TPM_MS_RESTRICT_APPROVE, a TPM_HMAC structure, generated by the TPM, signaling a valid signature over restrictTicket
14	4	11S	4	UINT32	encDataSize	The size of the encData parameter
15	<>	12S	<>	BYTE[]	encData	The encrypted entity that is to be modified.
16	4			TPM_AUTHHANDLE	parentAuthHandle	The authorization session handle used for the parent key.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
17	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with parentAuthHandle
18	1	4H1	1	BOOL	continueAuthSession	Continue use flag for parent session
19	20		20	TPM_AUTHDATA	parentAuth	HMAC key: parentKey.usageAuth.

2096Outgoing 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	HMAC key: parentKey.usageAuth.

2097Description

2098The TPM does not check the PCR values when migrating values locked to a PCR.

2099Actions

21001. Validate that parentAuth authorizes the use of the key pointed to by parentHandle.

21012. The TPM MAY verify that migrationType == migrationKeyAuth -> migrationScheme and
2102 return TPM_BAD_MODE on error.

2103 a. The TPM MAY ignore migrationType.

21043. Verify that parentHandle-> keyFlags-> migratable == FALSE

21054. Create d1 by decrypting encData using the key pointed to by parentHandle.

21065. Verify that the digest within migrationKeyAuth is legal for this TPM and public key

21076. Verify that d1 -> payload == TPM_PT_MIGRATE_RESTRICTED or
2108 TPM_PT_MIGRATE_EXTERNAL

21097. Verify that the migration authorities in msaList are authorized to migrate this key

2110 a. Create M2 a TPM_CMK_MIGAUTH structure

2111 i. Set M2 -> msaDigest to SHA-1[msaList]

2112 ii. Set M2 -> pubKeyDigest to pubSourceKeyDigest

2113 b. Verify that d1 -> migrationAuth == HMAC(M2) using tpmProof as the secret and
2114 return error TPM_MA_AUTHORITY on mismatch

21158. If migrationKeyAuth -> migrationScheme == TPM_MS_RESTRICT_MIGRATE

2116 a. Verify that intended migration destination is an MA:

2117 i. For one of n=1 to n=(msaList -> MSAlist), verify that SHA-1[migrationKeyAuth
2118 -> migrationKey] == msaList -> migAuthDigest[n]

2119 b. Validate that the MA key is the correct type

2120 i. Validate that migrationKeyAuth -> migrationKey -> algorithmParms ->
2121 algorithmID == TPM_ALG_RSA

2122 ii. Validate that migrationKeyAuth -> migrationKey -> algorithmParms ->
2123 encScheme is an encryption scheme supported by the TPM

2124 iii. Validate that migrationKeyAuth -> migrationKey ->algorithmParms ->
2125 sigScheme is TPM_SS_NONE

2126 c. The TPM MAY validate that restrictTicketSize is zero.

2127 d. The TPM MAY validate that sigTicketSize is zero.

21289. else If migrationKeyAuth -> migrationScheme == TPM_MS_RESTRICT_APPROVE

2129 a. Verify that the intended migration destination has been approved by the MSA:

2130 i. Verify that for one of the n=1 to n=(msaList -> MSAlist) values of msaList ->
2131 migAuthDigest[n], sigTicket == HMAC (V1) using tpmProof as the secret where V1
2132 is a TPM_CMK_SIGTICKET structure such that:

2133 (1) V1 -> verKeyDigest = msaList -> migAuthDigest[n]

2134 (2) V1 -> signedData = SHA-1[restrictTicket]

2135 ii. If [restrictTicket -> destinationKeyDigest] != SHA-1[migrationKeyAuth ->
2136 migrationKey], return error TPM_MA_DESTINATION

2137 iii. If [restrictTicket -> sourceKeyDigest] != pubSourceKeyDigest, return error
2138 TPM_MA_SOURCE

213910. Else return with error TPM_BAD_PARAMETER.

214011. Build two bytes array, K1 and K2, using d1:

2141 a. K1 = TPM_STORE_ASYMKEY.privKey[0..19]
2142 (TPM_STORE_ASYMKEY.privKey.keyLength + 16 bytes of
2143 TPM_STORE_ASYMKEY.privKey.key), sizeof(K1) = 20

2144 b. K2 = TPM_STORE_ASYMKEY.privKey[20..131] (position 16-127 of
2145 TPM_STORE_ASYMKEY . privKey.key), sizeof(K2) = 112

214612. Build M1 a TPM_MIGRATE_ASYMKEY structure

2147 a. TPM_MIGRATE_ASYMKEY.payload = TPM_PT_CMK_MIGRATE

2148 b. TPM_MIGRATE_ASYMKEY.usageAuth = TPM_STORE_ASYMKEY.usageAuth

2149 c. TPM_MIGRATE_ASYMKEY.pubDataDigest = TPM_STORE_ASYMKEY. pubDataDigest

2150 d. TPM_MIGRATE_ASYMKEY.partPrivKeyLen = 112 – 127.

2151 e. TPM_MIGRATE_ASYMKEY.partPrivKey = K2

215213. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP
2153 encoding of m using OAEP parameters m, pHash, and seed

2154 a. m is the previously created M1

b. pHash = SHA-1(SHA-1[msaList] || pubSourceKeyDigest)

c. seed = s1 = the previously created K1

14. 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

15. Create x1 by XOR of o1 with r1

16. Copy r1 into the output field “random”

17. Encrypt x1 with the migrationKeyAuth-> migrationKey

11.10 TPM_CMK_ConvertMigration

Start of informative comment:

TPM_CMK_ConvertMigration completes the migration of certified migration blobs.

This command takes a certified migration blob and creates a normal wrapped blob with payload type TPM_PT_MIGRATE_EXTERNAL. 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 TPM_LoadKey command.

TPM_CMK_ConvertMigration checks that one of the MAs implicitly listed in the migrationAuth of the target key has approved migration of the target key to the destination (parent) key, and that the settings (flags etc.) in the target key are those of a CMK.

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_ConvertMigration
4	4			TPM_KEY_HANDLE	parentHandle	Handle of a loaded key that can decrypt keys.
5	60	2S	60	TPM_CMK_AUTH	restrictTicket	The digests of public keys belonging to the Migration Authority, the destination parent key and the key-to-be-migrated.
6	20	3S	20	TPM_HMAC	sigTicket	A signature ticket, generated by the TPM, signaling a valid signature over restrictTicket
7	<>	4S	<>	TPM_KEY12	migratedKey	The public key of the key-to-be-migrated. The private portion MUST be TPM_MIGRATE_ASYMKEY properly XOR'd
8	4	5S	4	UINT32	msaListSize	The size of the msaList parameter, which is a variable length TPM_MSA_COMPOSITE structure
9	<>	6S	<>	TPM_MSA_COMPOSITE	msaList	One or more digests of public keys belonging to migration authorities
10	4	7S	4	UINT32	randomSize	Size of random
11	<>	8S	<>	BYTE [ ]	random	Random value used to hide key data.
12	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle.
		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 session handle
15	20			TPM_AUTHDATA	parentAuth	Authorization HMAC: 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_ConvertMigration
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	Authorization HMAC key .usageAuth

Action

1. Validate the AuthData 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 migratedKey -> encData area using the key in parentHandle
4. Create o1 by XOR d1 and random parameter
5. Create m1 a TPM_MIGRATE_ASYMKEY, seed and pHash by OAEP decoding o1
6. Create migratedPubKey a TPM_PUBKEY structure corresponding to migratedKey
   a. Verify that pHash == SHA-1(SHA-1[msaList] || SHA-1(migratedPubKey)
7. Create k1 by combining seed and the TPM_MIGRATE_ASYMKEY -> partPrivKey field
8. Create d2 a TPM_STORE_ASYMKEY structure.
   a. Set the TPM_STORE_ASYMKEY -> privKey field to k1
   b. Set d2 -> usageAuth to m1 -> usageAuth
   c. Set d2 -> pubDataDigest to m1 -> pubDataDigest
9. Verify that parentHandle-> keyFlags -> migratable == FALSE
10. Verify that m1 -> payload == TPM_PT_CMK_MIGRATE then set d2-> payload = TPM_PT_MIGRATE_EXTERNAL
11. Verify that for one of the n=1 to n=(msaList -> MSAlist) values of msaList -> migAuthDigest[n] sigTicket == HMAC (V1) using tpmProof as the secret where V1 is a TPM_CMK_SIGTICKET structure such that:
    a. V1 -> verKeyDigest = msaList -> migAuthDigest[n]
    b. V1 -> signedData = SHA-1[restrictTicket]
12. Create parentPubKey, a TPM_PUBKEY structure corresponding to parentHandle

220213. If [restrictTicket -> destinationKeyDigest] != SHA-1(parentPubKey), return error 2203 TPM_MA_DESTINATION

220414. Verify that migratedKey is corresponding to d2

220515. If migratedKey -> keyFlags -> migratable is FALSE, and return error 2206 TPM_INVALID_KEYUSAGE

220716. If migratedKey -> keyFlags -> migrateAuthority is FALSE, return error 2208 TPM_INVALID_KEYUSAGE

220917. If [restrictTicket -> sourceKeyDigest] != SHA-1(migratedPubKey), return error 2210 TPM_MA_SOURCE

221118. Create M2 a TPM_CMK_MIGAUTH structure

2212 a. Set M2 -> msaDigest to SHA-1[msaList]

2213 b. Set M2 -> pubKeyDigest to SHA-1[migratedPubKey]

221419. Set d2 -> migrationAuth = HMAC(M2) using tpmProof as the secret

221520. Create outData using the key in parentHandle to perform the encryption

12. Maintenance Functions (optional)

Start of informative comment:

When a maintenance archive is created with generateRandom FALSE, the maintenance blob is XOR encrypted with the owner authorization before encryption with the maintenance public key. This prevents the manufacturer from obtaining plaintext data. The receiving TPM must have the same owner authorization as the sending TPM in order to XOR decrypt the archive.

When generateRandom is TRUE, the maintenance blob is XOR encrypted with random data, which is also returned. This permits someone trusted by the Owner to load the maintenance archive into the replacement platform in the absence of the Owner and manufacturer, without the Owner having to reveal information about his auth value. The receiving and sending TPM's may have different owner authorizations. The random data is transferred from the sending TPM owner to the receiving TPM owner out of band, so the maintenance blob remains hidden from the manufacturer.

This is a typical maintenance sequence:

1. Manufacturer:
   - generates maintenance key pair
   - gives public key to TPM1 owner
2. TPM1: TPM_LoadManuMaintPub
   - load maintenance public key
3. TPM1: TPM_CreateMaintenanceArchive
   - XOR encrypt with owner auth or random
   - encrypt the maintenance archive with maintenance public key
4. TPM2:
   - Take ownership
   - Create and activate an AIK
   - Certify the SRK with the AIK, proving that the SRK came from a legitimate TPM
4. Manufacturer:
   - decrypt maintenance archive with maintenance private key
   - (still XOR encrypted with owner auth or random)
   - validate the TPM2 SRK certification
   - encrypt the maintenance archive with TPM2 SRK public key
5. TPM2: TPM_LoadMaintenanceArchive
   - decrypt the maintenance archive with SRK private key
   - XOR decrypt with owner auth or random

End of informative comment.

2252

12.1 TPM_CreateMaintenanceArchive

Start of informative comment:

This command creates the maintenance archive. 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 session handle used for owner authentication.
		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 session handle
8	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	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 session digest for the returned parameters. HMAC key: ownerAuth.

Actions

Upon authorization being confirmed this command does the following:

22631. Validates that the TPM_PERMANENT_FLAGS -> allowMaintenance is TRUE. If it is
2264 FALSE, the TPM SHALL return TPM_DISABLED_CMD and exit this capability.

22652. Validates the TPM Owner AuthData.

22663. If the value of TPM_PERMANENT_DATA -> manuMaintPub is zero, the TPM MUST
2267 return the error code TPM_KEYNOTFOUND

22684. Build a1 a TPM_KEY structure using the SRK. The encData field is not a normal
2269 TPM_STORE_ASYMKEY structure but rather a TPM_MIGRATE_ASYMKEY structure built
2270 using the following actions.

22715. Build a TPM_STORE_PRIVKEY structure from the SRK. This privKey element should be
2272 132 bytes long for a 2K RSA key.

22736. Create k1 and k2 by splitting the privKey element created in step 4 into 2 parts. k1 is
2274 the first 20 bytes of privKey, k2 contains the remainder of privKey.

22757. Build m1 by creating and filling in a TPM_MIGRATE_ASYMKEY structure

2276 a. m1 -> usageAuth is set to TPM_PERMANENT_DATA -> tpmProof

2277 b. m1 -> pubDataDigest is set to the digest value of the SRK fields from step 4

2278 c. m1 -> payload is set to TPM_PT_MAINT

2279 d. m1 -> partPrivKey is set to k2

22808. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP
2281 encoding of m using OAEP parameters of

2282 a. m = TPM_MIGRATE_ASYMKEY structure (step 7)

2283 b. pHash = TPM_PERMANENT_DATA -> ownerAuth

2284 c. seed = s1 = k1 (step 6)

22859. If generateRandom = TRUE

2286 a. Create r1 by obtaining values from the TPM RNG. The size of r1 MUST be the same
2287 size as o1. Set random parameter to r1

228810. If generateRandom = FALSE

2289 a. Create r1 by applying MGF1 to the TPM Owner AuthData. The size of r1 MUST be the
2290 same size as o1. Set randomSize to 0.

229111. Create x1 by XOR of o1 with r1

229212. Encrypt x1 with the manuMaintPub key using the TPM_ES_RSAESOAEP_SHA1_MGF1
2293 encryption scheme.

229413. Set a1 -> encData to the encryption of x1

229514. Set TPM_PERMANENT_FLAGS -> maintenanceDone to TRUE

229615. Return a1 in the archive parameter

2297 12.2 TPM_LoadMaintenanceArchive

2298 **Start of informative comment:**

2299 This command loads in a Maintenance archive that has been massaged by the
2300 manufacturer to load into another TPM.

2301 If the maintenance archive was created using the owner authorization for XOR encryption,
2302 the current owner authorization must be used for decryption. The owner authorization does
2303 not change.

2304 If the maintenance archive was created using random data for the XOR encryption, the
2305 vendor specific arguments must include the random data. The owner authorization may
2306 change.

2307 **End of informative comment.**

2308 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_LoadMaintenanceArchive
4	4	2S	4	UINT32	archiveSize	Sice of the encrypted archive
5	<>	3S	<>	BYTE[]	archive	Encrypted key archive
				…	…	Vendor specific arguments
-	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
			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 session handle
--	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

2309

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 session digest for the returned parameters. HMAC key: ownerAuth, the original value and not the new auth value

Description

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 AuthData 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 AuthData
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 section 12.1.
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.

23365. Execute the actions of TPM_OwnerClear.

23376. Process the maintenance information

2338 a. Update the SRK

2339 i. Set the SRK usageAuth to be the same as the source TPM owner's AuthData

2340 b. Update TPM_PERMANENT_DATA -> tpmProof

2341 c. Update TPM_PERMANENT_DATA -> ownerAuth

23427. Set TPM_PERMANENT_FLAGS -> maintenanceDone to TRUE

2343

12.3 TPM_KillMaintenanceFeature

Start of informative comment:

The TPM_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 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_KillMaintenanceFeature
4	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		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 session handle
7	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	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	HMAC key: ownerAuth.

Actions

1. Validate the TPM Owner AuthData
2. Set the TPM_PERMANENT_FLAGS.allowMaintenance flag to FALSE.

12.4 TPM_LoadManuMaintPub

Start of informative comment:

The TPM_LoadManuMaintPub command loads the manufacturer's public key for use in the maintenance process. The command installs manuMaintPub in PERMANENT 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 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadManuMaintPub
4	20	2S	20	TPM_NONCE	antiReplay	AntiReplay and validation nonce
5	<>	3S	<>	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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadManuMaintPub
4	20	3S	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. That is, this ordinal can only be run once, and used to either load the key or load a NULL key.

2383**Actions**

2384The first valid TPM_LoadManuMaintPub command received by a TPM SHALL

23851. Store the parameter pubKey as TPM_PERMANENT_DATA -> manuMaintPub.

23862. Set checksum to SHA-1 of (pubKey || antiReplay)

23873. Export the checksum

23884. Subsequent calls to TPM_LoadManuMaintPub SHALL return code 2389 TPM_DISABLED_CMD.

12.5 TPM_ReadManuMaintPub

Start of informative comment:

The TPM_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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadManuMaintPub
4	20	2S	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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadManuMaintPub
4	20	3S	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_ReadManuMaintPub command SHALL

1. Create "checksum" by concatenating data to form (TPM_PERMANENT_DATA -> manuMaintPub ||antiReplay) and passing the concatenated data through SHA-1.
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.

The exposure of the SHA-1 processing is a convenience to platforms in a mode that do not have sufficient memory to perform SHA-1 themselves. As such, the use of SHA-1 is restrictive on the TPM.

The TPM may not allow any other types of processing during the execution of a SHA-1 session. There is only one SHA-1 session active on a TPM. The exclusivity of a SHA-1 context is due to the relatively large volatile buffer it requires in order to hold the intermediate results between the SHA-1 context commands. This buffer can be in contradiction to other command needs.

After the execution of TPM_SHA1Start, and prior to TPM_SHA1Complete or TPM_SHA1CompleteExtend, the receipt of any command other than TPM_SHA1Update will cause the invalidation of the SHA-1 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SHA1Start
4	4	3S	4	UINT32	maxNumBytes	Maximum number of bytes that can be sent to TPM_SHA1Update. Must be a multiple of 64 bytes.

Description

1. 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.

24322. After receipt of TPM_SHA1Start, and prior to the receipt of TPM_SHA1Complete or
2433 TPM_SHA1CompleteExtend, receipt of any command other than TPM_SHA1Update
2434 invalidates the SHA-1 session.

2435 a. If the command received is TPM_ExecuteTransport, the SHA-1 session invalidation is
2436 based on the wrapped command, not the TPM_ExecuteTransport ordinal.

2437 b. A SHA-1 thread (start, update, complete) MUST take place either completely outside
2438 a transport session or completely within a single transport session.

2439 13.2 TPM_SHA1Update

2440**Start of informative comment:**

2441This capability inputs complete blocks of data into a pending SHA-1 digest. At the end of 2442the process, the digest remains pending.

2443**End of informative comment.**

2444 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SHA1Update
4	4	2S	4	UINT32	numBytes	The number of bytes in hashData. Must be a multiple of 64 bytes.
5	<>	3S	<>	BYTE [ ]	hashData	Bytes to be hashed

2445 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SHA1Update

2446 Description

2447This command SHALL incorporate complete blocks of data into the digest of an existing 2448SHA-1 thread. Only integral numbers of complete blocks (64 bytes each) can be processed.

2449 Actions

24501. If there is no existing SHA-1 thread, return TPM_SHA_THREAD

24512. If numBytes is not a multiple of 64

2452 a. Return TPM_SHA_ERROR

2453 b. The TPM MAY terminate the SHA-1 thread

24543. If numBytes is greater than maxNumBytes returned by TPM_SHA1Start

2455 a. Return TPM_SHA_ERROR

2456 b. The TPM MAY terminate the SHA-1 thread

24574. Incorporate hashData into the digest of the existing SHA-1 thread.

2458

2459 13.3 TPM_SHA1Complete

2460**Start of informative comment:**

2461This capability terminates a pending SHA-1 calculation.

2462**End of informative comment.**

2463 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SHA1Complete
4	4	2S	4	UINT32	hashDataSize	Number of bytes in hashData, MUST be 64 or less
5	<>	3S	<>	BYTE [ ]	hashData	Final bytes to be hashed

2464 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SHA1Complete
4	20	3S	20	TPM_DIGEST	hashValue	The output of the SHA-1 hash.

2465 Description

2466This command SHALL incorporate a partial or complete block of data into the digest of an 2467existing SHA-1 thread, and terminate that thread. hashDataSize MAY have values in the 2468range of 0 through 64, inclusive.

2469If the SHA-1 thread has received no bytes the TPM SHALL calculate the SHA-1 of the empty 2470buffer.

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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SHA1CompleteExtend
4	4	2S	4	TPM_PCRINDEX	pcrNum	Index of the PCR to be modified
5	4	3S	4	UINT32	hashDataSize	Number of bytes in hashData, MUST be 64 or less
6	<>	4S	<>	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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SHA1CompleteExtend
4	20	3S	20	TPM_DIGEST	hashValue	The output of the SHA-1 hash.
5	20	4S	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 SHA-1 session. hashDataSize MAY have values in the range of 0 through 64, inclusive.

The SHA-1 session MUST terminate even if the command returns an error, e.g. TPM_BAD_LOCALITY.

Actions

1. Validate that pcrNum represents a legal PCR number. On error, return TPM_BADINDEX.
2. Map V1 to TPM_STANY_DATA
3. Map L1 to V1 -> localityModifier

24908. If the current locality, held in L1, is not selected in TPM_PERMANENT_DATA -> pcrAttrib
2491 [pcrNum]. pcrExtendLocal, return TPM_BAD_LOCALITY

24929. Create H1 the TPM_DIGEST of the SHA-1 session ensuring that hashData, if any, is
2493 added to the SHA-1 session

249410. Perform the actions of TPM_Extend using H1 as the data and pcrNum as the PCR to
2495 extend

13.5 TPM_Sign

Start of informative comment:

The Sign command signs data and returns the resulting digital signature.

The TPM does not allow TPM_Sign with a TPM_KEY_IDENTITY (AIK) because TPM_Sign can sign arbitrary data and could be used to fake a quote. (This could have been relaxed to allow TPM_Sign with an AIK if the signature scheme is _INFO For an _INFO key, the metadata prevents TPM_Sign from faking a quote.)

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_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 session 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 session handle
10	20			TPM_AUTHDATA	privAuth	The authorization session 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: 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 session digest for the returned parameters. HMAC key: key.usageAuth

2506**Description**

2507The TPM MUST support all values of areaToSignSize that are legal for the defined signature
2508scheme and key size. The maximum value of areaToSignSize is determined by the defined
2509signature scheme and key size.

2510In the case of PKCS1v15_SHA1 the areaToSignSize MUST be TPM_DIGEST (the hash size of
2511a SHA-1 operation - see 8.5.1 TPM_SS_RSASSAPKCS1v15_SHA1). In the case of
2512PKCS1v15_DER the maximum size of areaToSign is k-11 octets, where k is limited by the
2513key size (see TPM_SS_RSASSAPKCS1v15_DER).

2514**Actions**

25151. The TPM validates the AuthData to use the key pointed to by keyHandle.

25162. If the areaToSignSize is 0 the TPM returns TPM_BAD_PARAMETER.

25173. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING or TPM_KEY_LEGACY, if not
2518 return the error code TPM_INVALID_KEYUSAGE

25194. The TPM verifies that the signature scheme and key size can properly sign the
2520 areaToSign parameter.

25215. If signature scheme is TPM_SS_RSASSAPKCS1v15_SHA1 then

2522 a. Validate that areaToSignSize is 20 return TPM_BAD_PARAMETER on error

2523 b. Set S1 to areaToSign

25246. Else if signature scheme is TPM_SS_RSASSAPKCS1v15_DER then

2525 a. Validate that areaToSignSize is at least 11 bytes less than the key size, return
2526 TPM_BAD_PARAMETER on error

2527 b. Set S1 to areaToSign

25287. else if signature scheme is TPM_SS_RSASSAPKCS1v15_INFO then

2529 a. Create S2 a TPM_SIGN_INFO structure

2530 b. Set S2 -> fixed to “SIGN”

2531 c. Set S2 -> replay to nonceOdd

2532 i. If nonceOdd is not present due to an unauthorized command return
2533 TPM_BAD_PARAMETER

2534 d. Set S2 -> dataLen to areaToSignSize

2535 e. Set S2 -> data to areaToSign

2536 f. Set S1 to the SHA-1(S2)

25378. Else return TPM_INVALID_KEYUSAGE

25389. The TPM computes the signature, sig, using the key referenced by keyHandle using S1
2539 as the value to sign

254010. Return the computed signature in Sig

13.6 TPM_GetRandom

Start of informative comment:

TPM_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 more than the number of bytes available is an 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetRandom.
4	4	2S	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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_GetRandom.
4	4	3S	4	UINT32	randomBytesSize	The number of bytes returned
5	<>	4S	<>	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 bytesRequested.
3. Set randomBytes to the next randomBytesSize bytes from the RNG

2556 13.7 TPM_StirRandom

2557**Start of informative comment:**

2558TPM_StirRandom adds entropy to the RNG state.

2559**End of informative comment.**

2560 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_StirRandom
4	4	2S	4	UINT32	dataSize	Number of bytes of input
5	<>	3S	<>	BYTE[ ]	inData	Data to add entropy to RNG state

2561 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_StirRandom

2562 Actions

25631. If dataSize is not less than 256 bytes, the TPM MAY return TPM_BAD_PARAMETER.

25642. 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 one key to certify the public portion of another key.

A TPM identity key may be used to certify non-migratable keys but is not permitted to certify migratory keys or certified migration 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 entity 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 the TPM_CERTIFY_INFO or TPM_CERTIFY_INFO2 on the output is based on which PCRs and what localities the certified key is restricted to. A key to be certified that does not have locality restrictions and which uses no PCRs greater than PCR #15 will cause this command to return and sign a TPM_CERTIFY_INFO structure, which provides compatibility with V1.1 TPMs.

When this command is run to certify all other keys (those that use PCR #16 or higher, as well as those limited by locality in any way), it will return and sign a TPM_CERTIFY_INFO2 structure.

TPM_CertifyKey does not support the case where (a) the certifying key requires a usage authorization to be provided but (b) the key-to-be-certified does not. In such cases, TPM_CertifyKey2 must be used. TPM_CertifyKey cannot be used to certify CMKs.

If a command tag (in the parameter array) specifies only one authorisation session, then the TPM convention is that the first session listed is ignored (authDataUsage must be TPM_AUTH_NEVER for this key) and the incoming session data is used for the second auth session in the list. In TPM_CertifyKey, the first session is the certifying key and the second session is the key-to-be-certified. In TPM_CertifyKey2, the first session is the key-to-be-certified and the second session is the certifying key.

The key handles of both the certifying key and the key to be certified are not included in the HMAC protecting the command. This permits key handle virtualization (swapping of keys in and out of the TPM that results in different key handles while at the same time maintaining key identifiers of upper layer software). In environments where the interface to the TPM is accessible by other parties, the key handles not being protected allows an attacker to change the handle of the key to be certified. This can be avoided by processing this command within a transport session and making sure that antiReplay indeed contains a nonce.

End of informative comment.

2603Incoming 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_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 session 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 session handle
10	20			TPM_AUTHDATA	certAuth	The authorization session digest for inputs and certHandle. HMAC key: certKey.auth.
11	4			TPM_AUTHHANDLE	keyAuthHandle	The authorization session 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 session handle
14	20			TPM_AUTHDATA	keyAuth	The authorization session 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 or TPM_CERTIFY_INFO2 structure 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 signature of certifyInfo
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 session 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	continueKeySession	Continue use flag for target key session
12	20			TPM_AUTHDATA	keyAuth	The authorization session 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. Verify command and key AuthData values:

   a. If tag is TPM_TAG_RQU_AUTH2_COMMAND

      i. The TPM verifies the AuthData in certAuthHandle provides authorization to use the key pointed to by certHandle, return TPM_AUTHFAIL on error

      ii. The TPM verifies the AuthData in keyAuthHandle provides authorization to use the key pointed to by keyHandle, return TPM_AUTH2FAIL on error

   b. else if tag is TPM_TAG_RQU_AUTH1_COMMAND

      i. Verify that authDataUsage is TPM_AUTH_NEVER for the key referenced by certHandle, return TPM_AUTHFAIL on error.

      ii. The TPM verifies the AuthData in keyAuthHandle provides authorization to use the key pointed to by keyHandle, return TPM_AUTHFAIL on error

   c. else if tag is TPM_TAG_RQU_COMMAND

      i. Verify that authDataUsage is TPM_AUTH_NEVER for the key referenced by certHandle, return TPM_AUTHFAIL on error.

2622 ii. Verify that authDataUsage is TPM_AUTH_NEVER or
2623 TPM_NO_READ_PUBKEY_AUTH for the key referenced by keyHandle, return
2624 TPM_AUTHFAIL on error.

26253. If keyHandle -> payload is not TPM_PT_ASYM, return TPM_INVALID_KEYUSAGE.

26264. If the key pointed to by certHandle is an identity key (certHandle -> keyUsage is
2627 TPM_KEY_IDENTITY)

2628 a. If keyHandle -> keyFlags -> migratable is TRUE return TPM_MIGRATEFAIL

26295. Validate that certHandle -> keyUsage is TPM_KEY_SIGN, TPM_KEY_IDENTITY or
2630 TPM_KEY_LEGACY, if not return TPM_INVALID_KEYUSAGE

26316. Validate that keyHandle -> keyUsage is TPM_KEY_SIGN, TPM_KEY_STORAGE,
2632 TPM_KEY_IDENTITY, TPM_KEY_BIND or TPM_KEY_LEGACY, if not return
2633 TPM_INVALID_KEYUSAGE

26347. If keyHandle -> digestAtRelease requires the use of PCRs 16 or higher to calculate or if
2635 keyHandle -> localityAtRelease is not 0x1F

2636 a. Set V1 to 1.2

26378. Else

2638 a. Set V1 to 1.1

26399. If keyHandle -> pcrInfoSize is not 0

2640 a. If keyHandle -> keyFlags has pcrIgnoredOnRead set to FALSE

2641 i. Create a digestAtRelease according to the specified TPM_STCLEAR_DATA ->
2642 PCR registers and compare to keyHandle -> digestAtRelease and if a mismatch
2643 return TPM_WRONGPCRVAL

2644 ii. If specified validate any locality requests on error TPM_BAD_LOCALITY

2645 b. If V1 is 1.1

2646 i. Create C1 a TPM_CERTIFY_INFO structure

2647 ii. Fill in C1 with the information from the key pointed to by keyHandle

2648 iii. The TPM MUST set c1 -> pcrInfoSize to 44.

2649 iv. The TPM MUST set c1 -> pcrInfo to a TPM_PCR_INFO structure properly filled
2650 out using the information from keyHandle.

2651 v. The TPM MUST set c1 -> digestAtCreation to 20 bytes of 0x00.

2652 c. Else

2653 i. Create C1 a TPM_CERTIFY_INFO2 structure

2654 ii. Fill in C1 with the information from the key pointed to by keyHandle

2655 iii. Set C1 -> pcrInfoSize to the size of an appropriate TPM_PCR_INFO_SHORT
2656 structure.

2657 iv. Set C1 -> pcrInfo to a properly filled out TPM_PCR_INFO_SHORT structure,
2658 using the information from keyHandle.

2659 v. Set C1 -> migrationAuthoritySize to 0

266010. Else

2661 a. Create C1 a TPM_CERTIFY_INFO structure

2662 b. Fill in C1 with the information from the key pointed to by keyHandle

2663 c. The TPM MUST set c1 -> pcrInfoSize to 0

266411. Create TPM_DIGEST H1 which is the SHA-1 hash of keyHandle -> pubKey -> key. Note
2665 that <key> is the actual public modulus, and does not include any structure formatting.

266612. Set C1 -> pubKeyDigest to H1

266713. The TPM copies the antiReplay parameter to c1 -> data.

266814. The TPM sets certifyInfo to C1.

266915. The TPM creates m1, a message digest formed by taking the SHA-1 of c1.

2670 a. The TPM then computes a signature using certHandle -> sigScheme. The resulting
2671 signed blob is returned in outData.

13.9 TPM_CertifyKey2

Start of informative comment:

This command is based on TPM_CertifyKey, but includes the ability to certify a Certifiable Migration Key (CMK), which requires extra input parameters.

TPM_CertifyKey2 always produces a TPM_CERTIFY_INFO2 structure.

TPM_CertifyKey2 does not support the case where (a) the key-to-be-certified requires a usage authorization to be provided but (b) the certifying key does not.

If a command tag (in the parameter array) specifies only one authorisation session, then the TPM convention is that the first session listed is ignored (authDataUsage must be TPM_NO_READ_PUBKEY_AUTH or TPM_AUTH_NEVER for this key) and the incoming session data is used for the second auth session in the list. In TPM_CertifyKey2, the first session is the key to be certified and the second session is the certifying key.

The key handles of both the certifying key and the key to be certified are not included in the HMAC protecting the command. This permits key handle virtualization (swapping of keys in and out of the TPM that results in different key handles while at the same time maintaining key identifiers of upper layer software). In environments where the interface to the TPM is accessible by other parties, the key handles not being protected allows an attacker to change the handle of the key to be certified. This can be avoided by processing this command within a transport session and making sure that antiReplay indeed contains a nonce.

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_CertifyKey2
4	4			TPM_KEY_HANDLE	keyHandle	Handle of the key to be certified.
5	4			TPM_KEY_HANDLE	certHandle	Handle of the key to be used to certify the key.
6	20	2S	20	TPM_DIGEST	migrationPubDigest	The digest of a TPM_MSA_COMPOSITE structure, containing at least one 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	keyAuthHandle	The authorization session handle used for the key to be signed.
		2H1	20	TPM_NONCE	keylastNonceEven	Even nonce previously generated by TPM
9	20	3H1	20	TPM_NONCE	keynonceOdd	Nonce generated by system associated with keyAuthHandle
10	1	4H1	1	BOOL	continueKeySession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	keyAuth	The authorization session digest for the inputs and key to be signed. HMAC key: key.usageAuth.
12	4			TPM_AUTHHANDLE	certAuthHandle	The authorization session handle used for certHandle.
		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 certAuthHandle
14	1	4H2	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
15	20			TPM_AUTHDATA	certAuth	Authorization HMAC key: certKey.auth.

2694Outgoing 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_INFO2	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	keyNonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	keyNonceOdd	Nonce generated by system associated with certAuthHandle
8	1	4H1	1	BOOL	keyContinueAuthSession	Continue use flag for cert key session
9	20		20	TPM_AUTHDATA	keyResAuth	Authorization HMAC key: keyHandle -> auth.
10	20	2H2	20	TPM_NONCE	certNonceEven	Even nonce newly generated by TPM
		3H2	20	TPM_NONCE	AuthLastNonceOdd	Nonce generated by system associated with certAuthHandle
11	1	4H2	1	BOOL	CertContinueAuthSession	Continue use flag for cert key session
12	20		20	TPM_AUTHDATA	certResAuth	Authorization HMAC key: certHandle -> auth.

2695Actions

26961. The TPM validates that the key pointed to by certHandle has a signature scheme of 2697 TPM_SS_RSASSAPKCS1v15_SHA1 or TPM_SS_RSASSAPKCS1v15_INFO

26982. Verify command and key AuthData values:

2699 a. If tag is TPM_TAG_RQU_AUTH2_COMMAND

2700 i. The TPM verifies the AuthData in keyAuthHandle provides authorization to 2701 use the key pointed to by keyHandle, return TPM_AUTHFAIL on error

2702 ii. The TPM verifies the AuthData in certAuthHandle provides authorization to 2703 use the key pointed to by certHandle, return TPM_AUTH2FAIL on error

2704 b. else if tag is TPM_TAG_RQU_AUTH1_COMMAND

2705 i. Verify that authDataUsage is TPM_AUTH_NEVER or 2706 TPM_NO_READ_PUBKEY_AUTH for the key referenced by keyHandle, return 2707 TPM_AUTHFAIL on error

2708 ii. The TPM verifies the AuthData in certAuthHandle provides authorization to 2709 use the key pointed to by certHandle, return TPM_AUTH2FAIL on error

2710 c. else if tag is TPM_TAG_RQU_COMMAND

2711 i. Verify that authDataUsage is TPM_AUTH_NEVER or 2712 TPM_NO_READ_PUBKEY_AUTH for the key referenced by keyHandle, return 2713 TPM_AUTHFAIL on error

ii. Verify that authDataUsage is TPM_AUTH_NEVER for the key referenced by certHandle, return TPM_AUTHFAIL on error.

3. If the key pointed to by certHandle is an identity key (certHandle -> keyUsage is TPM_KEY_IDENTITY)

a. If keyHandle -> keyFlags -> migratable is TRUE and [keyHandle -> keyFlags-> migrateAuthority is FALSE or (keyHandle -> payload != TPM_PT_MIGRATE_RESTRICTED and keyHandle -> payload != TPM_PT_MIGRATE_EXTERNAL)] return TPM_MIGRATEFAIL

4. Validate that certHandle -> keyUsage is TPM_KEY_SIGNING, TPM_KEY_IDENTITY or TPM_KEY_LEGACY, if not return TPM_INVALID_KEYUSAGE

5. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING, TPM_KEY_STORAGE, TPM_KEY_IDENTITY, TPM_KEY_BIND or TPM_KEY_LEGACY, if not return TPM_INVALID_KEYUSAGE

6. The TPM SHALL create a c1 a TPM_CERTIFY_INFO2 structure from the key pointed to by keyHandle

7. Create TPM_DIGEST H1 which is the SHA-1 hash of keyHandle -> pubKey -> key. Note that <key> is the actual public modulus, and does not include any structure formatting.

8. Set C1 -> pubKeyDigest to H1

9. Copy the antiReplay parameter to c1 -> data

10. Copy other keyHandle parameters into C1

11. If keyHandle -> payload == TPM_PT_MIGRATE_RESTRICTED or TPM_PT_MIGRATE_EXTERNAL

a. create thisPubKey, a TPM_PUBKEY structure containing the public key, algorithm and parameters corresponding to keyHandle

b. Verify that the migration authorization is valid for this key

i. Create M2 a TPM_CMK_MIGAUTH structure

ii. Set M2 -> msaDigest to migrationPubDigest

iii. Set M2 -> pubkeyDigest to SHA-1[thisPubKey]

iv. Verify that [keyHandle -> migrationAuth] == HMAC(M2) signed by using tpmProof as the secret and return error TPM_MA_SOURCE on mismatch

c. Set C1 -> migrationAuthority = SHA-1(migrationPubDigest || keyHandle -> payload)

d. if keyHandle -> payload == TPM_PT_MIGRATE_RESTRICTED

i. Set C1 -> payloadType = TPM_PT_MIGRATE_RESTRICTED

e. if keyHandle -> payload == TPM_PT_MIGRATE_EXTERNAL

i. Set C1 -> payloadType = TPM_PT_MIGRATE_EXTERNAL

12. Else

a. set C1 -> migrationAuthority = NULL

b. set C1 -> migrationAuthoritySize =0

c. Set C1 -> payloadType = TPM_PT_ASYM

13. If keyHandle -> pcrInfoSize is not 0

a. The TPM MUST set c1 -> pcrInfoSize to match the pcrInfoSize from the keyHandle key.

b. The TPM MUST set c1 -> pcrInfo to match the pcrInfo from the keyHandle key

c. If keyHandle -> keyFlags has pcrIgnoredOnRead set to FALSE

i. Create a digestAtRelease according to the specified TPM_STCLEAR_DATA -> 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

14. Else

a. The TPM MUST set c1 -> pcrInfoSize to 0

15. The TPM creates m1, a message digest formed by taking the SHA-1 of c1

a. The TPM then computes a signature using certHandle -> sigScheme. The resulting signed blob is returned in outData

2767 14. Endorsement Key Handling

2768**Start of informative comment:**

2769There are two create EK commands. The first matches the 1.1 functionality. The second
2770provides the mechanism to enable revokeEK.

2771The TPM and platform manufacturer decide on the inclusion or exclusion of the ability to
2772execute revokeEK.

2773The restriction to have the TPM generate the EK does not remove the manufacturing option
2774to "squirt" the EK. During manufacturing, the TPM does not enforce all protections or
2775requirements; hence, the restriction on only TPM generation of the EK is also not in force.

2776**End of informative comment.**

27771. A TPM SHALL NOT install an EK unless generated on the TPM by execution of
2778 TPM_CreateEndorsementKeyPair or TPM_CreateRevocableEK

2779 14.1 TPM_CreateEndorsementKeyPair

2780**Start of informative comment:**

2781This command creates the TPM endorsement key. It returns a failure code if an 2782endorsement key already exists.

2783**End of informative comment.**

2784 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateEndorsementKeyPair
4	20	2S	20	TPM_NONCE	antiReplay	Arbitrary data
5	<>	3S	<>	TPM_KEY_PARMS	keyInfo	Information about key to be created, this includes all algorithm parameters

2785 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateEndorsementKeyPair
4	<>	3S	<>	TPM_PUBKEY	pubEndorsementKey	The public endorsement key
5	20	4S	20	TPM_DIGEST	checksum	Hash of pubEndorsementKey and antiReplay

2786 Actions

27871. If an EK already exists, return TPM_DISABLED_CMD

27882. Validate the keyInfo parameters for the key description

2789 a. If the algorithm type is RSA the key length MUST be a minimum of 2048. For
2790 interoperability the key length SHOULD be 2048

2791 b. If the algorithm type is other than RSA the strength provided by the key MUST be
2792 comparable to RSA 2048

2793 c. The other parameters of keyInfo (encScheme, sigScheme, etc.) are ignored.

27943. Create a key pair called the “endorsement key pair” using a TPM-protected capability.
2795 The type and size of key are that indicated by keyInfo. Set encScheme to
2796 TPM_ES_RSAESOAEP_SHA1_MGF1.

27974. Create checksum by performing SHA-1 on the concatenation of (PUBEK || antiReplay)

27985. Store the PRIVEK

27996. Create TPM_PERMANENT_DATA -> tpmDAASeed from the TPM RNG

28007. Create TPM_PERMANENT_DATA -> daaProof from the TPM RNG

28018. Create TPM_PERMANENT_DATA -> daaBlobKey from the TPM RNG

28029. Set TPM_PERMANENT_FLAGS -> CEKPUsed to TRUE

280310. Set TPM_PERMANENT_FLAGS -> enableRevokeEK to FALSE

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 AuthData value to reset the EK will be generated by the TPM, and the new AuthData value itself if it is not to be generated by the TPM. The output parameter is the new AuthData 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

2834**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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateRevocableEK
4	20	2S	20	TPM_NONCE	antiReplay	Arbitrary data
5	<>	3S	<>	TPM_KEY_PARMS	keyInfo	Information about key to be created, this includes all algorithm parameters
6	1	4S	1	BOOL	generateReset	If TRUE use TPM RNG to generate EKreset. If FALSE use the passed value inputEKreset
7	20	5S	20	TPM_NONCE	inputEKreset	The authorization value to be used with TPM_RevokeTrust if generateReset==FALSE, else the parameter is present but ignored

2835**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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateRevocableEK
4	<>	3S	<>	TPM_PUBKEY	pubEndorsementKey	The public endorsement key
5	20	4S	20	TPM_DIGEST	checksum	Hash of pubEndorsementKey and antiReplay
6	20	5S	20	TPM_NONCE	outputEKreset	The AuthData value to use TPM_RevokeTrust

2836**Actions**

28371. If an EK already exists, return TPM_DISABLED_CMD

28382. Perform the actions of TPM_CreateEndorsementKeyPair, if any errors return with error

28393. Set TPM_PERMANENT_FLAGS -> enableRevokeEK to TRUE

2840 a. If generateReset is TRUE then

2841 i. Set TPM_PERMANENT_DATA -> EKreset to the next value from the TPM RNG

2842 b. Else

2843 i. Set TPM_PERMANENT_DATA -> EKreset to inputEKreset

28444. Return PUBEK, checksum and Ekreset

28455. The outputEKreset AuthData is sent in the clear. There is no uniqueness on the TPM 2846 available to actually perform encryption or use an encrypted channel. The assumption is 2847 that this operation is occurring in a controlled environment and sending the value in the 2848 clear is acceptable.

2849 14.3 TPM_RevokeTrust

2850**Start of informative comment:**

2851This command clears the EK and sets the TPM back to a pure default state. The generation
2852of the AuthData value occurs during the generation of the EK. It is the responsibility of the
2853EK generator to properly protect and disseminate the RevokeTrust AuthData.

2854**End of informative comment.**

2855This is an optional command

2856 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_RevokeTrust
4	20	2S	20	TPM_NONCE	EKReset	The value that will be matched to EK Reset

2857 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_RevokeTrust

2858 Actions

28591. The TPM MUST validate that TPM_PERMANENT_FLAGS -> enableRevokeEK is TRUE,
2860 return TPM_PERMANENTEK on error

28612. The TPM MUST validate that the EKReset matches TPM_PERMANENT_DATA -> EKReset
2862 return TPM_AUTHFAIL on error.

28633. Ensure that physical presence is being asserted

28644. Perform the actions of TPM_OwnerClear (excepting the command authentication)

2865 a. NV items with the pubInfo -> nvIndex D value set MUST be deleted. This changes the
2866 TPM_OwnerClear handling of the same NV areas

2867 b. Set TPM_PERMANENT_FLAGS -> nvLocked to FALSE

28685. Invalidate TPM_PERMANENT_DATA -> tpmDAASeed

28696. Invalidate TPM_PERMANENT_DATA -> daaProof

28707. Invalidate TPM_PERMANENT_DATA -> daaBlobKey

28718. Invalidate the EK and any internal state associated with the EK

2872 14.4 TPM_ReadPubek

2873**Start of informative comment:**

2874Return the endorsement key public portion. This value should have controls placed upon 2875access, as it is a privacy sensitive value.

2876The readPubek flag is set to FALSE by TPM_TakeOwnership and set to TRUE by 2877TPM_OwnerClear, thus mirroring if a TPM Owner is present.

2878**End of informative comment.**

2879 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadPubek
4	20	2S	20	TPM_NONCE	antiReplay	Arbitrary data

2880 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadPubek
4	<>	3S	<>	TPM_PUBKEY	pubEndorsementKey	The public endorsement key
5	20	4S	20	TPM_DIGEST	checksum	Hash of pubEndorsementKey and antiReplay

2881 Description

2882This command returns the PUBEK.

2883 Actions

2884The TPM_ReadPubek command SHALL

28851. If TPM_PERMANENT_FLAGS -> readPubek is FALSE return TPM_DISABLED_CMD

28862. If no EK is present the TPM MUST return TPM_NO_ENDORSEMENT

28873. Create checksum by performing SHA-1 on the concatenation of (pubEndorsementKey || 2888 antiReplay).

28894. Export the PUBEK and checksum.

2890 14.5 TPM_OwnerReadInternalPub

2891 **Start of informative comment:**

2892 A TPM Owner authorized command that returns the public portion of the EK or SRK.

2893 The keyHandle parameter is included in the incoming session authorization to prevent
2894 alteration of the value, causing a different key to be read. Unlike most key handles, which
2895 can be mapped by higher layer software, this key handle has only two fixed values.

2896 **End of informative comment.**

2897 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 session handle used for owner authentication.
		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 session handle
8	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

2898 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 session digest for the returned parameters. HMAC key: ownerAuth.

2899 Actions

29001. Validate the parameters and TPM Owner AuthData for this command

29012. If keyHandle is TPM_KH_EK

2902 a. Set publicPortion to PUBEK

29033. Else If keyHandle is TPM_KH_SRK

2904 a. Set publicPortion to the TPM_PUBKEY of the SRK

29054. Else return TPM_BAD_PARAMETER

29065. Export the public key of the referenced key

2907 15. Identity Creation and Activation

2908 15.1 TPM_MakeIdentity

2909 **Start of informative comment:**

2910 Generate a new Attestation Identity Key (AIK).

2911 labelPrivCADigest identifies the privacy CA that the owner expects to be the target CA for
2912 the AIK. The selection is not enforced by the TPM. It is advisory only. It is included
2913 because the TSS cannot be trusted to send the AIK to the correct privacy CA. The privacy
2914 CA can use this parameter to validate that it is the target privacy CA and label intended by
2915 the TPM owner at the time the key was created. The label can be used to indicate an
2916 application purpose.

2917 **End of informative comment.**

2918 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 AuthData for the new identity
5	20	3S	20	TPM_CHOSENID_HASH	labelPrivCADigest	The digest of the identity label and privacy CA chosen for the AIK
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 session 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 session digest for the inputs and the SRK. HMAC key: srk.usageAuth.
11	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication. 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 session digest for inputs and owner. HMAC key: ownerAuth.

2919Outgoing 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	Continue use flag. Fixed value of FALSE
9	20			TPM_AUTHDATA	srkAuth	The authorization session 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	Continue use flag. Fixed value of FALSE
12	20		20	TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

2920Description

2921The public key of the new TPM identity SHALL be identityPubKey. The private key of the 2922new TPM identity SHALL be tpm_signature_key.

2923Properties 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.

2924
2925This capability also generates a TPM_KEY containing the tpm_signature_key.

2926If identityPubKey is stored on a platform it SHALL exist only in storage to which access is 2927controlled and is available to authorized entities.

2928The signing of TPM_ID_CONTENTS is not an authorized use of the key. The 2929TPM_PCR_INFO_xxx "…AtRelease" values are not validated.

2930Actions

2931A Trusted Platform Module that receives a valid TPM_MakeIdentity command SHALL do the 2932following:

29331. Validate the idKeyParams parameters for the key description

2934 a. If the algorithm type is RSA, the key length MUST be a minimum of 2048 and MUST
2935 use the default exponent. For interoperability the key length SHOULD be 2048.

2936 b. If the algorithm type is other than RSA the strength provided by the key MUST be
2937 comparable to RSA 2048

2938 c. If the TPM is not designed to create a key of the requested type, return the error code
2939 TPM_BAD_KEY_PROPERTY

2940 d. If TPM_PERMANENT_FLAGS -> FIPS is TRUE then

2941 i. If authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS

29422. Use authHandle to verify that the Owner authorized all TPM_MakeIdentity input
2943 parameters.

29443. Use srkAuthHandle to verify that the SRK owner authorized all TPM_MakeIdentity input
2945 parameters.

29464. Verify that idKeyParams -> keyUsage is TPM_KEY_IDENTITY. If it is not, return
2947 TPM_INVALID_KEYUSAGE

29485. Verify that idKeyParams -> keyFlags -> migratable is FALSE. If it is not, return
2949 TPM_INVALID_KEYUSAGE

29506. Create a1 by decrypting identityAuth according to the ADIP indicated by authHandle.

29517. Set continueAuthSession and continueSRKSession to FALSE.

29528. Determine the structure version

2953 a. If idKeyParams -> tag is TPM_TAG_KEY12

2954 i. Set V1 to 2

2955 ii. Create idKey a TPM_KEY12 structure using idKeyParams as the default values
2956 for the structure

2957 b. If idKeyParams -> ver is 1.1

2958 i. Set V1 to 1

2959 ii. Create idKey a TPM_KEY structure using idKeyParams as the default values
2960 for the structure

29619. Set the digestAtCreation values for pcrInfo

2962 a. For TPM_PCR_INFO_LONG include the locality of the current command

296310. Create an asymmetric key pair (identityPubKey and tpm_signature_key) using a TPM-
2964 protected capability, in accordance with the algorithm specified in idKeyParams

296511. Ensure that the AuthData information in A1 is properly stored in the idKey as
2966 usageAuth.

296712. Attach identityPubKey and tpm_signature_key to idKey

296813. Set idKey -> migrationAuth to TPM_PERMANENT_DATA-> tpmProof

296914. Ensure that all TPM_PAYLOAD_TYPE structures identify this key as TPM_PT_ASYM

297015. Encrypt the private portion of idKey using the SRK as the parent key

297116. Create a TPM_IDENTITY_CONTENTS structure named idContents using
2972 labelPrivCADigest and the information from idKey

297317. Sign idContents using tpm_signature_key and TPM_SS_RSASSAPKCS1v15_SHA1. Store
2974 the result in identityBinding.

2975 15.2 TPM_ActivateIdentity

2976**Start of informative comment:**

2977The purpose of TPM_ActivateIdentity is to twofold. The first purpose is to obtain assurance 2978that the credential in the TPM_SYM_CA_ATTESTATION is for this TPM. The second purpose 2979is to obtain the session key used to encrypt the TPM_IDENTITY_CREDENTIAL.

2980This is an extension to the 1.1 functionality of TPM_ActivateIdentity. The blob sent to from 2981the CA can be in the 1.1 format or the 1.2 format. The TPM determines the type from the 2982size or version information in the blob.

2983TPM_ActivateIdentity checks that the symmetric session key corresponds to a TPM-identity 2984before releasing that session key.

2985Only the Owner of the TPM has the privilege of activating a TPM identity. The Owner is 2986required to authorize the TPM_ActivateIdentity command. The owner may authorize the 2987command using either the TPM_OIAP or TPM_OSAP authorization protocols.

2988The creator of the ActivateIdentity package can specify if any PCR values are to be checked 2989before releasing the session key.

2990**End of informative comment.**

2991 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	idKeyHandle	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 session 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 session digest for the inputs and ID key. HMAC key: idKey.usageAuth.
11	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		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 session handle
14	20		20	TPM_AUTHDATA	ownerAuth	The authorization session 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 session 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 session digest for the returned parameters. HMAC key: ownerAuth.

Description

1. The command TPM_ActivateIdentity activates a TPM identity created using the command TPM_MakeIdentity.
2. 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.
3. 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 AuthData to execute the command and all of the incoming parameters.
2. Using the idKeyAuthHandle, validate the AuthData 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 idKeyHandle -> keyUsage is TPM_KEY_IDENTITY. Return TPM_BAD_PARAMETER on mismatch
4. Create H1 the digest of a TPM_PUBKEY derived from 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. Validate that B1 -> ekType is TPM_EK_TYPE_ACTIVATE, return TPM_BAD_TYPE if not.
b. Assign A1 as a TPM_EK_BLOB_ACTIVATE structure from B1 -> blob
c. Compare H1 to A1 -> idDigest on mismatch return TPM_BAD_PARAMETER
d. If A1 -> pcrSelection is not NULL
    i. Compute a composite hash C1 using the PCR selection A1 -> pcrSelection
    ii. Compare C1 to A1 -> pcrInfo->digestAtRelease and return TPM_WRONGPCRVAL on a mismatch
    e. If A1 -> pcrInfo specifies a locality ensure that the appropriate locality has been asserted, return TPM_BAD_LOCALITY on error
f. 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 TPM_STCLEAR_DATA -> PCR
   a. TPM_Extend
   b. TPM_SHA1CompleteExtend
   c. TPM_Startup
   d. TPM_PCR_Reset

3046 16.1 TPM_Extend

3047**Start of informative comment:**

3048This adds a new measurement to a PCR

3049**End of informative comment.**

3050 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Extend.
4	4	2S	4	TPM_PCRINDEX	pcrNum	The PCR to be updated.
5	20	3S	20	TPM_DIGEST	inDigest	The 160 bit value representing the event to be recorded.

3051 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Extend.
4	20	3S	20	TPM_PCRVALUE	outDigest	The PCR value after execution of the command.

3052 Description

3053Add a measurement value to a PCR

3054 Actions

30551. Validate that pcrNum represents a legal PCR number. On error, return TPM_BADINDEX.

30562. Map L1 to TPM_STANY_FLAGS -> localityModifier

30573. Map P1 to TPM_PERMANENT_DATA -> pcrAttrib [pcrNum]. pcrExtendLocal

30584. If, for the value of L1, the corresponding bit is not set in the bit map P1, return 3059 TPM_BAD_LOCALITY

30605. Create c1 by concatenating (TPM_STCLEAR_DATA -> PCR[pcrNum] || inDigest). This 3061 takes the current PCR value and concatenates the inDigest parameter.

30626. Create h1 by performing a SHA-1 digest of c1.

30637. Store h1 to TPM_STCLEAR_DATA -> PCR[pcrNum]

30648. If TPM_PERMANENT_FLAGS -> disable is TRUE or TPM_STCLEAR_FLAGS -> deactivated 3065 is TRUE

3066 a. Set outDigest to 20 bytes of 0x00

30679. Else

3068 a. Set outDigest to h1

3069 16.2 TPM_PCRRead

3070**Start of informative comment:**

3071The TPM_PCRRead operation provides non-cryptographic reporting of the contents of a 3072named PCR.

3073**End of informative comment.**

3074 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PCRRead.
4	4	2S	4	TPM_PCRINDEX	pcrIndex	Index of the PCR to be read

3075 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PCRRead.
4	20	3S	20	TPM_PCRVALUE	outDigest	The current contents of the named PCR

3076 Description

3077The TPM_PCRRead operation returns the current contents of the named register to the 3078caller.

3079 Actions

30801. Validate that pcrIndex represents a legal PCR number. On error, return 3081 TPM_BADINDEX.

30822. Set outDigest to TPM_STCLEAR_DATA -> PCR[pcrIndex]

30833. Return TPM_SUCCESS

3084

3085 16.3 TPM_Quote

3086**Start of informative comment:**

3087The TPM_Quote operation provides cryptographic reporting of PCR values. A loaded key is
3088required for operation. TPM_Quote uses a key to sign a statement that names the current
3089value of a chosen PCR and externally supplied data (which may be a nonce supplied by a
3090Challenger).

3091The term "ExternalData" is used because an important use of TPM_Quote is to provide a
3092digital signature on arbitrary data, where the signature includes the PCR values of the
3093platform at time of signing. Hence the "ExternalData" is not just for anti-replay purposes,
3094although it is (of course) used for that purpose in an integrity challenge.

3095TPM_Quote should not use a TPM_KEY_SIGNING, because there is no way for the remote
3096party to tell whether TPM_Quote or TPM_Sign created the signature. The exception is a
3097TPM_KEY_SIGNING key with the _INFO signature scheme, because the metadata
3098differentiates TPM_Sign from TPM_Quote.

3099**End of informative comment.**

3100 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_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	externalData	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 session 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 session handle
10	20			TPM_AUTHDATA	privAuth	The authorization session digest for inputs and keyHandle. HMAC key: key -> usageAuth.

3101**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_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 session digest for the returned parameters. HMAC key: Key -> usageAuth.

3102**Actions**

31031. The TPM MUST validate the AuthData to use the key pointed to by keyHandle.

31042. Validate that keyHandle -> sigScheme is TPM_SS_RSASSAPKCS1v15_SHA1 or
3105 TPM_SS_RSASSAPKCS1v15_INFO, if not return TPM_INAPPROPRIATE_SIG.

31063. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING, TPM_KEY_IDENTITY, or
3107 TPM_KEY_LEGACY, if not return TPM_INVALID_KEYUSAGE

31084. Validate targetPCR

3109 a. targetPCR is a valid TPM_PCR_SELECTION structure

3110 b. On errors return TPM_INVALID_PCR_INFO

31115. Create H1 a SHA-1 hash of a TPM_PCR_COMPOSITE using the TPM_STCLEAR_DATA ->
3112 PCR indicated by targetPCR -> pcrSelect

31136. Create Q1 a TPM_QUOTE_INFO structure

3114 a. Set Q1 -> version to 1.1.0.0

3115 b. Set Q1 -> fixed to “QUOT”

3116 c. Set Q1 -> digestValue to H1

3117 d. Set Q1 -> externalData to externalData

31187. Sign SHA-1 hash of Q1 using keyHandle as the signature key

31198. Return the signature in sig

16.4 TPM_PCR_Reset

Start of informative comment:

For PCR with the pcrReset attribute set to TRUE, this command resets the PCR back to the default value, this mimics the actions of TPM_Init. The PCR may have restrictions as to which locality can perform the reset operation.

Sending a null pcrSelection results in an error is due to the requirement that the command actually do something. If pcrSelection is null there are no PCR to reset and the command would then do nothing.

For PCR that are resettable, the presence of a Trusted Operating System (TOS) can change the behavior of TPM_PCR_Reset. The following pseudo code shows how the behavior changes

```
At TPM_Startup
    If TPM_PCR_ATTRIBUTES->pcrReset is FALSE
        Set PCR to 0x00…00
    Else
        Set PCR to 0xFF…FF
At TPM_PCR_Reset
    If TPM_PCR_ATTRIBUTES->pcrReset is TRUE
        If TOSPresent
                Set PCR to 0x00…00
        Else
                Set PCR to 0xFF…FF
    Else
        Return error
```

The above pseudocode is for example only, for the details of a specific platform, the reader must review the platform specific specification. The purpose of the above pseudocode is to show that both pcrReset and the TOSPresent bit control the value in use to when the PCR resets.

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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PCR_Reset
4	<>	2S	<>	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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_PCR_Reset

Description

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. is a valid TPM_PCR_SELECTION structure
   b. pcrSelection -> pcrSelect is non-zero
   c. On errors return TPM_INVALID_PCR_INFO
2. Map L1 to TPM_STANY_FLAGS -> localityModifier
3. For each PCR selected perform the following
   a. If TPM_PERMANENT_DATA -> pcrAttrib[pcrIndex].pcrReset is FALSE, return TPM_NOTRESETABLE
   b. If, for the value L1, the corresponding bit is clear in the bit map TPM_PERMANENT_DATA -> pcrAttrib[pcrIndex].pcrResetLocal, return TPM_NOTLOCAL
4. For each PCR selected perform the following
   a. The PCR MAY only reset to 0x00…00 or 0xFF…FF
   b. The logic to determine which value to use MUST be described by a platform specific specification

16.5 TPM_Quote2

Start of informative comment:

The TPM_Quote2 operation provides cryptographic reporting of PCR values. A loaded key is required for operation. TPM_Quote2 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_Quote2 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.

TPM_Quote2 differs from TPM_Quote in that TPM_Quote2 uses TPM_PCR_INFO_SHORT to hold information relative to the PCR registers. TPM_PCR_INFO_SHORT includes locality information to provide the requestor a more complete view of the current platform configuration.

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_Quote2
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	externalData	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	1	4S	1	BOOL	addVersion	When TRUE add TPM_CAP_VERSION_INFO to the output
8	4			TPM_AUTHHANDLE	authHandle	The authorization session 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 session handle
11	20			TPM_AUTHDATA	privAuth	The authorization session digest for inputs and keyHandle. HMAC key: key -> usageAuth.

3186**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_Quote2
4	<>	3S	<>	TPM_PCR_INFO_SHORT	pcrData	The value created and signed for the quote
5	4	4S	4	UINT32	versionInfoSize	Size of the version info
6	<>	5S	<>	TPM_CAP_VERSION_INFO	versionInfo	The version info
7	4	6S	4	UINT32	sigSize	The used size of the output area for the signature
8	<>	7S	<>	BYTE[ ]	sig	The signed data blob.
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 session digest for the returned parameters. HMAC key: Key -> usageAuth.

3187**Actions**

31881. The TPM MUST validate the AuthData to use the key pointed to by keyHandle.

31892. Validate that keyHandle -> sigScheme is TPM_SS_RSASSAPKCS1v15_SHA1 or
3190 TPM_SS_RSASSAPKCS1v15_INFO, if not return TPM_INAPPROPRIATE_SIG.

31913. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING, TPM_KEY_IDENTITY, or
3192 TPM_KEY_LEGACY, if not return TPM_INVALID_KEYUSAGE

31934. Validate targetPCR is a valid TPM_PCR_SELECTION structure, on errors return
3194 TPM_INVALID_PCR_INFO

31955. Create H1 a SHA-1 hash of a TPM_PCR_COMPOSITE using the TPM_STCLEAR_DATA ->
3196 PCR[] indicated by targetPCR -> pcrSelect

31976. Create S1 a TPM_PCR_INFO_SHORT

3198 a. Set S1->pcrSelection to targetPCR

3199 b. Set S1->localityAtRelease to TPM_STANY_DATA -> localityModifier

3200 c. Set S1->digestAtRelease to H1

32017. Create Q1 a TPM_QUOTE_INFO2 structure

3202 a. Set Q1 -> fixed to “QUT2”

3203 b. Set Q1 -> infoShort to S1

3204 c. Set Q1 -> externalData to externalData

32058. If addVersion is TRUE

3206 a. Concatenate to Q1 a TPM_CAP_VERSION_INFO structure

3207 b. Set the output parameters for versionInfo

32089. Else

3209 a. Set versionInfoSize to 0

3210 b. Return no bytes in versionInfo

321110. Sign a SHA-1 hash of Q1 using keyHandle as the signature key

321211. Return the signature in sig

17. Changing AuthData

17.1 TPM_ChangeAuth

Start of informative comment:

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

This command cannot invalidate the old entity. Therefore, the authorization change is only effective if the application can guarantee that the old entity can be securely destroyed. If not, two valid entities will exist, one with the old and one with the new authorization secret.

If this command is delegated, the delegated party can expand its key use privileges. That party can create a copy of the key with known authorization, and it can then use the key without any ordinal restrictions.

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 parameters used for used encryption are generated from the authLastNonceEven (created during the OSAP session), nonceOdd, and the session shared secret.

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.

3231

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_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 AuthData for the entity
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 session 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 session digest for inputs and parentHandle. HMAC key: parentKey.usageAuth.
14	4			TPM_AUTHHANDLE	entityAuthHandle	The authorization session 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 session digest for the inputs and encrypted entity. HMAC key: entity.usageAuth.

3232Outgoing 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: 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 session 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	continueEntitySession	Continue use flag, fixed value of FALSE
11	20			TPM_AUTHDATA	entityAuth	The authorization session digest for the returned parameters and entity. HMAC key: entity.usageAuth, the original and not the new auth value

3233Description

32341. The parentAuthHandle session type MUST be TPM_PID_OSAP.

32352. In this capability, the SRK cannot be accessed as entityType TPM_ET_KEY, since the 3236 SRK is not wrapped by a parent key.

3237Actions

32381. Verify that entityType is one of TPM_ET_DATA, TPM_ET_KEY and return the error 3239 TPM_WRONG_ENTITYTYPE if not.

32402. Verify that parentAuthHandle session type is TPM_PID_OSAP return TPM_BAD_MODE 3241 on error

32423. Verify that entityAuthHandle session type is TPM_PID_OIAP return TPM_BAD_MODE on 3243 error

32444. If protocolID is not TPM_PID_ADCP, the TPM MUST return TPM_BAD_PARAMETER.

32455. The encData parameter MUST be the encData field from either the TPM_STORED_DATA 3246 or TPM_KEY structures.

32476. Create decryptAuth by decrypting newAuth according to the ADIP indicated by 3248 parentHandle.

32497. The TPM MUST validate the command using the AuthData in the parentAuth parameter

32508. Validate that parentHandle -> keyUsage is TPM_KEY_STORAGE, if not return 3251 TPM_INVALID_KEYUSAGE

32529. After parameter validation, the TPM creates b1 by decrypting encData using the key 3253 pointed to by parentHandle.

325410. The TPM MUST validate that b1 is a valid TPM structure, either a 3255 TPM_STORE_ASYMKEY or a TPM_SEALED_DATA

3256 a. Check the length and payload, return TPM_INVALID_STRUCTURE on any mismatch

3257 b. The TPM must validate the command using the authorization data entityAuth 3258 parameter. The HMAC key is TPM_STORE_ASYMKEY -> usageAuth or 3259 TPM_SEALED_DATA -> authData.

326011. The TPM replaces the AuthData for b1 with decryptAuth created above.

326112. The TPM encrypts b1 using the appropriate mechanism for the type using the 3262 parentKeyHandle to provide the key information.

326313. The TPM MUST enforce the destruction of both the parentAuthHandle and 3264 entityAuthHandle sessions.

3265 17.2 TPM_ChangeAuthOwner

3266**Start of informative comment:**

3267The TPM_ChangeAuthOwner command allows the owner of an entity to change the
3268AuthData for the TPM Owner or the SRK.

3269This command requires authorization from the current TPM Owner to execute.

3270TPM's targeted for an environment (e.g. a server) with long lasting sessions should not
3271invalidate all sessions.

3272**End of informative comment.**

3273 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 AuthData for the entity
6	2	4S	2	TPM_ENTITY_TYPE	entityType	The type of entity to be modified
7	4			TPM_AUTHHANDLE	ownerAuthHandle	The authorization session 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 session digest for inputs and ownerHandle. HMAC key: ownerAuth.

3274 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 session digest for the returned parameters and ownerHandle. HMAC key: ownerAuth, the original value and not the new auth value

Actions

1. The TPM MUST validate the command using the AuthData in the ownerAuth parameter
2. The ownerAuthHandle session type MUST be TPM_PID_OSAP
3. If protocolID is not TPM_PID_ADCP, the TPM MUST return TPM_BAD_PARAMETER.
4. Verify that entityType is either TPM_ET_OWNER or TPM_ET_SRK, and return the error TPM_WRONG_ENTITYTYPE if not.
5. Create decryptAuth by decrypting newAuth according to the ADIP indicated by ownerAuthHandle.
6. The TPM MUST enforce the destruction of the ownerAuthHandle session upon completion of this command (successful or unsuccessful). This includes setting continueAuthSession to FALSE
7. Set the AuthData for the indicated entity to decryptAuth
8. The TPM MUST invalidate all owner authorized OSAP and DSAP sessions, active or saved.
9. The TPM MAY invalidate all sessions, active or saved

3290 18. Authorization Sessions

3291 18.1 TPM_OIAP

3292 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OIAP.

3293 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OIAP.
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.

3294 Actions

32951. The TPM_OIAP command allows the creation of an authorization session handle and the 3296 tracking of the handle by the TPM. The TPM generates the handle and nonce.

32972. The TPM has an internal limit as to the number of handles that may be open at one 3298 time, so the request for a new handle may fail if there is insufficient space available.

32993. Internally the TPM will do the following:

3300 a. TPM allocates space to save handle, protocol identification, both nonces and any 3301 other information the TPM needs to manage the session.

3302 b. TPM generates authHandle and nonceEven, returns these to caller

33034. On each subsequent use of the OIAP session the TPM MUST generate a new nonceEven 3304 value.

33055. When TPM_OIAP is wrapped in an encrypted transport session, no input or output 3306 parameters are encrypted.

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 session 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 AuthData (SecretE, say) of the target entity. The entity and its secret must have been previously loaded into the TPM.
   a. If the command using the OIAP session requires owner authorization
      i. If TPM_STCLEAR_DATA -> ownerReference is TPM_KH_OWNER, the secret AuthData is TPM_PERMANENT_DATA -> ownerAuth
      ii. If TPM_STCLEAR_DATA -> ownerReference is pointing to a delegate row
      (1) Set R1 a row index to TPM_STCLEAR_DATA -> ownerReference
      (2) Set D1 a TPM_DELEGATE_TABLE_ROW to TPM_PERMANENT_DATA -> delegateTable -> delRow[R1]
      (3) Set the secret AuthData to D1 -> authValue
      (4) Validate the TPM_DELEGATE_PUBLIC D1 -> pub
      (a) Validate D1 -> pub -> permissions based on the command ordinal
      (b) Validate D1 -> pub -> pcrInfo based on the PCR values
5. The TPM SHALL perform a HMAC calculation using the entity secret data, ordinal, input command parameters and authorization parameters per Part 1 Object-Independent Authorization Protocol.
6. The TPM SHALL compare HM to the AuthData 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 Part 1 Object-Independent Authorization Protocol.

9. The TPM returns the return code, output parameters, authorization parameters and authorization session digest.

10. If the output continueUse flag is FALSE, then the TPM SHALL terminate the session. Future references to H will return an error.

11. Each time that access to an entity (e.g., key) is authorized using OIAP, the TPM MUST validate the TPM_PCR_INFO_xxx "…AtRelease" values if specified for the entity

a. The TPM SHOULD validate the values before using the shared secret to validate the command parameters. This prevents a dictionary attack on the shared secret when the values are invalid for the entity.

18.2 TPM_OSAP

Start of informative comment:

The TPM_OSAP command creates the authorization session 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OSAP.
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 session 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 AuthData for each command but not have to request the information or cache the actual AuthData.
4. When TPM_OSAP is wrapped in an encrypted transport session, no input or output parameters are encrypted.
5. If the owner pointer is pointing to a delegate row, the TPM internally MUST treat the OSAP session as a DSAP session

33806. TPM_ET_SRK or TPM_ET_KEYHANDLE with a value of TPM_KH_SRK MUST specify the 3381 SRK.

33827. If the entity is tied to PCR values, the PCR's are not validated during the TPM_OSAP 3383 ordinal session creation. The PCR's are validated when the OSAP session is used.

3384**Actions**

33851. The TPM creates S1 a storage area that keeps track of the information associated with 3386 the authorization.

33872. S1 MUST track the following information

3388 a. Protocol identification (i.e. TPM_PID_OSAP)

3389 b. nonceEven

3390 i. Initialized to the next value from the TPM RNG

3391 c. shared secret

3392 d. ADIP encryption scheme from TPM_ENTITY_TYPE entityType

3393 e. Any other internal TPM state the TPM needs to manage the session

33943. The TPM MUST create and MAY track the following information

3395 a. nonceEvenOSAP

3396 i. Initialized to the next value from the TPM RNG

33974. The TPM calculates the shared secret using an HMAC calculation. The key for the HMAC 3398 calculation is the secret AuthData assigned to the key handle identified by entityValue. 3399 The input to the HMAC calculation is the concatenation of nonces nonceEvenOSAP and 3400 nonceOddOSAP. The output of the HMAC calculation is the shared secret which is saved 3401 in the authorization area associated with authHandle

34025. Check if the ADIP encryption scheme specified by entityType is supported, if not return 3403 TPM_INAPPROPRIATE_ENC.

34046. If entityType = TPM_ET_KEYHANDLE

3405 a. The entity to authorize is a key held in the TPM. entityValue contains the keyHandle 3406 that holds the key.

3407 b. If entityValue is TPM_KH_OPERATOR return TPM_BAD_HANDLE

34087. else if entityType = TPM_ET_OWNER

3409 a. This value indicates that the entity is the TPM owner. entityValue is ignored

3410 b. The HMAC key is the secret pointed to by ownerReference (owner secret or delegated 3411 secret)

34128. else if entityType = TPM_ET_SRK

3413 a. The entity to authorize is the SRK. entityValue is ignored.

34149. else if entityType = TPM_ET_COUNTER

3415 a. The entity is a monotonic counter, entityValue contains the counter handle

341610. else if entityType = TPM_ET_NV

3417 a. The entity is a NV index, entityValue contains the NV index

341811. else return TPM_BAD_PARAMETER

341912. On each subsequent use of the OSAP session the TPM MUST generate a new nonce
3420 value.

342113. The TPM MUST ensure that OSAP shared secret is only available while the OSAP session
3422 is valid.

342314. The session MUST terminate upon any of the following conditions:

3424 a. The command that uses the session returns an error

3425 b. The resource is evicted from the TPM or otherwise invalidated

3426 c. The session is used in any command for which the shared secret is used to encrypt
3427 an input parameter (TPM_ENCAUTH)

3428 d. The TPM Owner is cleared

3429 e. TPM_ChangeAuthOwner is executed and this session is attached to the owner
3430 authorization

3431 f. The session explicitly terminated with continueAuth, TPM_Reset or
3432 TPM_FlushSpecific

3433 g. All OSAP sessions associated with the delegation table MUST be invalidated when
3434 any of the following commands execute:

3435 i. TPM_Delegate_Manage

3436 ii. TPM_Delegate_CreateOwnerDelegation with Increment==TRUE

3437 iii. TPM_Delegate_LoadOwnerDelegation

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 session 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 Part 1 Object-Specific Authorization Protocol.
5. The TPM SHALL compare HM1 to the AuthData 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 Part 1 Object-Specific Authorization Protocol.
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 (e.g., key) is authorized using OSAP, the TPM MUST
    a. ensure that the OSAP shared secret is that derived from the entity using TPM_OSAP
    b. validate the TPM_PCR_INFO_xxx "…AtRelease" values if specified for the entity

i. The TPM SHOULD validate the values before using the shared secret to validate the command parameters. This prevents a dictionary attack on the shared secret when the values are invalid for the entity.

3478 18.3 TPM_DSAP

3479**Start of informative comment:**

3480The TPM_DSAP command creates the authorization session handle using a delegated
3481AuthData value passed into the command as an encrypted blob or from the internal
3482delegation table. It can be used to start an authorization session for a user key or the
3483owner.

3484As in TPM_OSAP, it generates a shared secret and generates nonceEven and
3485nonceEvenOSAP.

3486**End of informative comment.**

3487 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DSAP.
4	2			TPM_ENTITY_TYPE	entityType	The type of delegation information to use
5	4			TPM_KEY_HANDLE	keyHandle	Key for which delegated authority corresponds, or 0 if delegated owner activity. Only relevant if entityValue equals TPM_DELEGATE_KEY_BLOB
6	20			TPM_NONCE	nonceOddDSAP	The nonce generated by the caller associated with the shared secret.
7	4			UINT32	entityValueSize	The size of entityValue.
8	<>	2S	<>	BYTE [ ]	entityValue	TPM_DELEGATE_KEY_BLOB or TPM_DELEGATE_OWNER_BLOB or index MUST not be empty If entityType is TPM_ET_DEL_ROW then entityValue is a TPM_DELEGATE_INDEX

3488 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DSAP.
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.

3489 Description

34901. The TPM_DSAP command allows the creation of an authorization session handle and the
3491 tracking of the handle by the TPM. The TPM generates the handle, nonceEven and
3492 nonceEvenOSAP.

34932. The TPM has an internal limit on the number of handles that may be open at one time, 3494 so the request for a new handle may fail if there is insufficient space available.

34953. The TPM_DSAP allows the binding of a delegated authorization to a specific entity. This 3496 allows the caller to continue to send in AuthData for each command but not have to 3497 request the information or cache the actual AuthData.

34984. Each ordinal that uses the DSAP session MUST validate that TPM_PERMANENT_DATA 3499 -> restrictDelegate does not restrict delegation, based on keyHandle -> keyUsage and 3500 keyHandle -> keyFlags, return TPM_INVALID_KEYUSAGE on error.

35015. On each subsequent use of the DSAP session the TPM MUST generate a new nonce 3502 value and check if the ordinal to be executed has delegation to execute. The TPM MUST 3503 ensure that the DSAP shared secret is only available while the DSAP session is valid.

35046. When TPM_DSAP is wrapped in an encrypted transport session

3505 a. For input the only parameter encrypted is entityValue

3506 b. For output no parameters are encrypted

35077. The DSAP session MUST terminate under any of the following conditions

3508 a. The command that uses the session returns an error

3509 b. If attached to a key, when the key is evicted from the TPM or otherwise invalidated

3510 c. The session is used in any command for which the shared secret is used to encrypt 3511 an input parameter (TPM_ENCAUTH)

3512 d. The TPM Owner is cleared

3513 e. TPM_ChangeAuthOwner is executed and this session is attached to the owner 3514 authorization

3515 f. The session explicitly terminated with continueAuth, TPM_Reset or 3516 TPM_FlushSpecific

3517 g. All DSAP sessions MUST be invalidated when any of the following commands 3518 execute:

3519 i. TPM_Delegate_CreateOwnerDelegation

3520 ii. When Increment is TRUE

3521 iii. TPM_Delegate_LoadOwnerDelegation

3522 iv. TPM_Delegate_Manage

3523entityType = TPM_ET_DEL_OWNER_BLOB

3524 The entityValue parameter contains an owner delegation blob structure.

3525entityType = TPM_ET_DEL_ROW

3526 The entityValue parameter contains a row number in the nv Delegation table which 3527 should be used for the AuthData value.

3528entityType = TPM_DEL_KEY_BLOB

3529 The entityValue parameter contains a key delegation blob structure.

Actions

1. If entityType == TPM_ET_DEL_OWNER_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 -> pub -> 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 all zeros
      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 decrypting B1 -> sensitiveArea using TPM_DELEGATE_KEY
   i. Validate S1 values
      i. S1 -> tag is TPM_TAG_DELEGATE_SENSITIVE
      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_BLOB
   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 -> pub -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate that row, return TPM_BADINDEX if not found

3566 d. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]

3567 e. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD

3568 f. Verify that K1 -> pub -> verificationCount equals FR -> verificationCount.

3569 g. Validate the integrity of the blob

3570 i. Copy K1 -> integrityDigest to H2

3571 ii. Set K1 -> integrityDigest to all zeros

3572 iii. Create H3 the HMAC of K1 using tpmProof as the secret

3573 iv. Compare H2 to H3 return TPM_AUTHFAIL on mismatch

3574 h. Validate that K1 -> pubKeyDigest identifies keyHandle, return TPM_KEYNOTFOUND
3575 on error

3576 i. Create S1 a TPM_DELEGATE_SENSITIVE by decrypting K1 -> sensitiveArea using
3577 TPM_DELEGATE_KEY

3578 j. Validate S1 values

3579 i. S1 -> tag is TPM_TAG_DELEGATE_SENSITIVE

3580 ii. Return TPM_BAD_DELEGATE on error

3581 k. Set A1 to S1 -> authValue

35824. Else return TPM_BAD_PARAMETER

35835. Generate a new authorization session handle and reserve space to save protocol
3584 identification, shared secret, pcrInfo, both nonces, ADIP encryption scheme, delegated
3585 permission bits and any other information the TPM needs to manage the session.

35866. Read two new values from the RNG to generate nonceEven and nonceEvenOSAP.

35877. The TPM calculates the shared secret using an HMAC calculation. The key for the HMAC
3588 calculation is A1. The input to the HMAC calculation is the concatenation of nonces
3589 nonceEvenOSAP and nonceOddOSAP. The output of the HMAC calculation is the shared
3590 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 to provide an owner delegation function for legacy code that does not itself support delegation. Normally, TPM_STCLEAR_DATA->ownerReference points to TPM_KH_OWNER, indicating that OIAP and OSAP sessions should use the owner authorization. This command allows ownerReference to point to an index in the delegation table, indicating that OIAP and OSAP sessions should use the delegation authorization.

In use, a TSS supporting delegation would create and load the owner delegation and set the owner pointer to that delegation. From then on, a legacy TSS application would use its OIAP and OSAP sessions with the delegated owner authorization.

Since this command is not authorized, the ownerReference is open to DoS attacks. Applications can attempt to recover from a failing owner authorization by resetting ownerReference to an appropriate value.

This command intentionally does not clear OSAP sessions. A TPM 1.1 application gets the benefit of owner delegation, while the original owner can use a pre-existing OSAP session with the actual 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_ 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_SetOwnerPointer
4	2	2S	2	TPM_ENTITY_TYPE	entityType	The type of entity in use
5	4	3S	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	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_SetOwnerPointer

Actions

1. Map TPM_STCLEAR_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 return TPM_BADINDEX

i. Set D1 to the delegation information in the row.

c. Locate D1 -> 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 B1->verificationCount equals FR -> verificationCount.

g. The TPM sets V1-> ownerReference to entityValue

h. 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_BAD_PARAMETER

3632 19. Delegation Commands

3633 19.1 TPM_Delegate_Manage

3634 **Start of informative comment:**

3635 TPM_Delegate_Manage is the fundamental process for managing the Family tables, 3636 including enabling/disabling Delegation for a selected Family. Normally 3637 TPM_Delegate_Manage must be executed at least once (to create Family tables for a 3638 particular family) before any other type of Delegation command in that family can succeed.

3639 TPM_Delegate_Manage is authorized by the TPM Owner if an Owner is installed, because 3640 changing a table is a privileged Owner operation. If no Owner is installed, 3641 TPM_Delegate_Manage requires no privilege to execute. This does not disenfranchise an 3642 Owner, since there is no Owner, and simplifies loading of tables during platform 3643 manufacture or on first-boot. Burn-out of TPM non-volatile storage by inappropriate use is 3644 mitigated by the TPM's normal limits on NV-writes in the absence of an Owner. Tables can 3645 be locked after loading, to prevent subsequent tampering, and only unlocked by the Owner, 3646 his delegate, or the act of removing the Owner (even if there is no Owner).

3647 TPM_Delegate_Manage command is customized by opCode:

3648 (1) TPM_FAMILY_ENABLE enables/disables use of a family and all the rows of the delegate 3649 table belonging to that family,

3650 (2) TPM_FAMILY_ADMIN can be used to prevent further management of the Tables until an 3651 Owner is installed, or until the Owner is removed from the TPM. (Note that the Physical 3652 Presence command TPM_ForceClear always enables further management, even if 3653 TPM_ForceClear is used when no Owner is installed.)

3654 (3) TPM_FAMILY_CREATE creates a new family. Sessions are invalidated even in this case 3655 because the lastFamilyID could wrap.

3656 (4) TPM_FAMILY_INVALIDATE invalidates an existing family. The TPM_SELFTEST_FAILED 3657 error code is returned because the familyRow has already been validated earlier. Failure 3658 here indicates a malfunction of the TPM.

3659 The rationale for Action 19.1 is that invalidating the family ID prevents the use of 3660 associated delegate rows, whether or not those delegate rows are themselves invalidated. 3661 Omitting the delegate row invalidation avoids an NV write.

3662 **End of informative comment.**

3663**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	opCode	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 opCode
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		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 session handle
11	20			TPM_AUTHDATA	ownerAuth	HMAC key: ownerAuth.

3664**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_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.

3665**Action**

36661. If opCode != TPM_FAMILY_CREATE

3667 a. Locate familyID in the TPM_FAMILY_TABLE and set familyRow to indicate row,
3668 return TPM_BADINDEX if not found

3669 b. Set FR, a TPM_FAMILY_TABLE_ENTRY, to TPM_FAMILY_TABLE. famTableRow
3670 [familyRow]

36712. If tag = TPM_TAG_RQU_AUTH1_COMMAND

3672 a. Validate the command and parameters using ownerAuth, return TPM_AUTHFAIL on
3673 error

b. If the command is delegated (authHandle session type is TPM_PID_DSAP or through ownerReference delegation)

i. If opCode = TPM_FAMILY_CREATE

(1) The TPM MUST ignore familyID

ii. Else

(1) Verify that the familyID associated with authHandle matches the familyID parameter, return TPM_DELEGATE_FAMILY on error

13. Else

a. If TPM_PERMANENT_DATA -> ownerAuth is valid, return TPM_AUTHFAIL

b. If opCode != TPM_FAMILY_CREATE and FR -> flags -> TPM_DELEGATE_ADMIN_LOCK is TRUE, return TPM_DELEGATE_LOCK

c. Validate max NV writes without an owner

i. Set NV1 to TPM_PERMANENT_DATA -> noOwnerNVWrite

ii. Increment NV1 by 1

iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITES

iv. Set TPM_PERMANENT_DATA -> noOwnerNVWrite to NV1

14. The TPM invalidates sessions

a. MUST invalidate all DSAP sessions

b. MUST invalidate all OSAP sessions associated with the delegation table

c. MUST set TPM_STCLEAR_DATA -> ownerReference to TPM_KH_OWNER

d. MAY invalidate any other session

15. If opCode == 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_PARAM_SIZE

c. Map F2 to a TPM_FAMILY_TABLE_ENTRY

i. Set F2 -> tag to TPM_TAG_FAMILY_TABLE_ENTRY

ii. Set F2 -> familyLabel to opData

d. Increment TPM_PERMANENT_DATA -> lastFamilyID by 1

e. Set F2 -> familyID = TPM_PERMANENT_DATA -> lastFamilyID

f. Set F2 -> verificationCount = 1

g. Set F2 -> flags -> TPM_FAMFLAG_ENABLED to FALSE

h. Set F2 -> flags -> TPM_DELEGATE_ADMIN_LOCK to FALSE

i. Set retDataSize = 4

3709 j. Set retData = F2 -> familyID
3710 k. Return TPM_SUCCESS
37116. If authHandle is of type DSAP then continueAuthSession MUST set to FALSE
37127. If opCode == TPM_FAMILY_ADMIN
3713 a. Validate that opDataSize == 1, and that opData is a Boolean value.
3714 b. Set (FR -> flags -> TPM_DELEGATE_ADMIN_LOCK) = opData
3715 c. Set retDataSize = 0
3716 d. Return TPM_SUCCESS
37178. else If opCode == TPM_FAMILY_ENABLE
3718 a. Validate that opDataSize == 1, and that opData is a Boolean value.
3719 b. Set FR -> flags-> TPM_FAMFLAG_ENABLED = opData
3720 c. Set retDataSize = 0
3721 d. Return TPM_SUCCESS
37229. else If opCode == TPM_FAMILY_INVALIDATE
3723 a. Invalidate all data associated with familyRow
3724 i. All data is all information pointed to by FR
3725 ii. return TPM_SELFTEST_FAILED on failure
3726 b. The TPM MAY invalidate delegate rows that contain the same familyID.
3727 c. Set retDataSize = 0
3728 d. Return TPM_SUCCESS
372910. Else return TPM_BAD_PARAMETER

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.

There is no check for appropriateness of the key's key usage against the key permission settings. If the key usage is incorrect, this command succeeds, but the delegated command will fail.

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

(TPM_Delegate_CreateOwnerDelegation must be used to delegate Owner privilege.)

The publicInfo -> familyID can specify a disabled family row. The family row is checked when the key delegation is used in a DSAP session, not when it is created.

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_CreateKeyDelegation.
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key.
5	<>	2S	<>	TPM_DELEGATE_PUBLIC	publicInfo	The public information necessary to fill in the blob
6	20	3S	20	TPM_ENCAUTH	delAuth	The encrypted new AuthData for the blob. The encryption key is the shared secret from the authorization session protocol.
7	4			TPM_AUTHHANDLE	authHandle	The authorization session 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	Ignored
10	20			TPM_AUTHDATA	privAuth	The authorization session digest that authorizes the use of keyHandle. HMAC key: key.usageAuth

3745**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_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. Fixed value of FALSE
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth

3746**Description**

37471. The use restrictions that may be present on the key pointed to by keyHandle are not 3748 enforced for this command. Stated another way, TPM_CreateKeyDelegation is not a use 3749 of the key.

3750**Action**

37511. Verify AuthData for the command and parameters using privAuth

37522. Locate publicInfo -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate 3753 row, return TPM_BADINDEX if not found

37543. If the key authentication is in fact a delegation, then the TPM SHALL validate the 3755 command and parameters using Delegation authorisation, then

3756 a. Validate that authHandle -> familyID equals publicInfo -> familyID return 3757 TPM_DELEGATE_FAMILY on error

3758 b. If TPM_FAMILY_TABLE.famTableRow[authHandle -> familyID] -> flags -> 3759 TPM_FAMFLAG_ENABLED is FALSE, return error TPM_DISABLED_CMD.

3760 c. Verify that the delegation bits in publicInfo do not grant more permissions then 3761 currently delegated. Otherwise return error TPM_AUTHFAIL

37624. Check that publicInfo -> delegateType is TPM_DEL_KEY_BITS

37635. Verify that authHandle indicates an OSAP or DSAP session return 3764 TPM_INVALID_AUTHHANDLE on error

37656. Create a1 by decrypting delAuth according to the ADIP indicated by authHandle.

37667. Create h1 the SHA-1 of TPM_STORE_PUBKEY structure of the key pointed to by 3767 keyHandle

37688. Create M1 a TPM_DELEGATE_SENSITIVE structure

3769 a. Set M1 -> tag to TPM_TAG_DELEGATE_SENSITIVE

3770 b. Set M1 -> authValue to a1

3771 c. The TPM MAY add additional information of a sensitive nature relative to the
3772 delegation

37739. Create M2 the encryption of M1 using TPM_DELEGATE_KEY

377410. Create P1 a TPM_DELEGATE_KEY_BLOB

3775 a. Set P1 -> tag to TPM_TAG_DELG_KEY_BLOB

3776 b. Set P1 -> pubKeyDigest to H1

3777 c. Set P1 -> pub to PublicInfo

3778 d. Set P1 -> pub -> verificationCount to familyRow -> verificationCount

3779 e. Set P1 -> integrityDigest to all zeros

3780 f. The TPM sets additionalArea and additionalAreaSize appropriate for this TPM. The
3781 information MAY include symmetric IV, symmetric mode of encryption and other data
3782 that allows the TPM to process the blob in the future.

3783 g. Set P1 -> sensitiveSize to the size of M2

3784 h. Set P1 -> sensitiveArea to M2

378511. Calculate H2 the HMAC of P1 using tpmProof as the secret

378612. Set P1 -> integrityDigest to H2

378713. Ignore continueAuthSession on input set continueAuthSession to FALSE on output

378814. Return P1 as blob

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_CreateOwnerDelegation (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	<>	3S	<>	TPM_DELEGATE_PUBLIC	publicInfo	The public parameters for the blob
6	20	4S	20	TPM_ENCAUTH	delAuth	The encrypted new AuthData for the blob. The encryption key is the shared secret from the OSAP protocol.
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle TPM Owner authentication
		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	Ignored
10	20			TPM_AUTHDATA	ownerAuth	The authorization session digest. HMAC key:ownerAuth

3810Outgoing 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_OWNER_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. Fixed value of FALSE
8	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth

3811Action

38121. The TPM SHALL authenticate the command using TPM Owner authentication. Return
3813 TPM_AUTHFAIL on failure.

38142. Locate publicInfo -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate
3815 the row return TPM_BADINDEX if not found

3816 a. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]

38173. If the TPM Owner authentication is in fact a delegation

3818 a. Validate that authHandle -> familyID equals publicInfo -> familyID return
3819 TPM_DELEGATE_FAMILY on error

3820 b. If FR -> flags -> TPM_FAMFLAG_ENABLED is FALSE, return error
3821 TPM_DISABLED_CMD.

3822 c. Verify that the delegation bits in publicInfo do not grant more permissions then
3823 currently delegated. Otherwise, return error TPM_AUTHFAIL.

38244. Check that publicInfo -> delegateType is TPM_DEL_OWNER_BITS

38255. Verify that authHandle indicates an OSAP or DSAP session return
3826 TPM_INVALID_AUTHHANDLE on error

38276. If increment == TRUE

3828 a. Increment FR -> verificationCount

3829 b. Set TPM_STCLEAR_DATA-> ownerReference to TPM_KH_OWNER

3830 c. The TPM invalidates sessions

3831 i. MUST invalidate all DSAP sessions

3832 ii. MUST invalidate all OSAP sessions associated with the delegation table

3833 iii. MAY invalidate any other session

38347. Create a1 by decrypting delAuth according to the ADIP indicated by authHandle.

38358. Create M1 a TPM_DELEGATE_SENSITIVE structure

3836 a. Set M1 -> tag to TPM_TAG_DELEGATE_SENSITIVE

3837 b. Set M1 -> authValue to a1

3838 c. Set other M1 fields as determined by the TPM vendor

38399. Create M2 the encryption of M1 using TPM_DELEGATE_KEY

384010. Create B1 a TPM_DELEGATE_OWNER_BLOB

3841 a. Set B1 -> tag to TPM_TAG_DELG_OWNER_BLOB

3842 b. Set B1 -> pub to publicInfo

3843 c. Set B1 -> sensitiveSize to the size of M2

3844 d. Set B1 -> sensitiveArea to M2

3845 e. Set B1 -> integrityDigest to all zeros

3846 f. Set B1 -> pub -> verificationCount to FR -> verificationCount

384711. The TPM sets additionalArea and additionalAreaSize appropriate for this TPM. The 3848 information MAY include symmetric IV, symmetric mode of encryption and other data 3849 that allows the TPM to process the blob in the future.

385012. Create H1 the HMAC of B1 using tpmProof as the secret

385113. Set B1 -> integrityDigest to H1

385214. Ignore continueAuthSession on input set continueAuthSession to FALSE on output

385315. Return B1 as blob

3854 19.4 TPM_Delegate_LoadOwnerDelegation

3855 **Start of informative comment:**

3856 This command loads a delegate table row blob into a non-volatile delegate table row.
3857 TPM_Delegate_LoadOwnerDelegation can be used during manufacturing or on first boot
3858 (when no Owner is installed), or after an Owner is installed. If an Owner is installed,
3859 TPM_Delegate_LoadOwnerDelegation requires Owner authorisation, and sensitive
3860 information must be encrypted.

3861 Burn-out of TPM non-volatile storage by inappropriate use is mitigated by the TPM's normal
3862 limits on NV-writes in the absence of an Owner. Tables can be locked after loading using
3863 TPM_Delegate_Manage, to prevent subsequent tampering.

3864 A management system outside the TPM is expected to manage the delegate table rows
3865 stored on the TPM, and can overwrite any previously stored data. There is no way to
3866 explicitly delete a delegation entry. A new entry can overwrite an invalid entry.

3867 This command cannot be used to load key delegation blobs into the TPM

3868 **End of informative comment.**

3869 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_LoadOwnerDelegation
4 4 3S 4 TPM_DELEGATE_INDEX index The index of the delegate ro	4	3S	4	TPM_DELEGATE_INDEX	index	The index of the delegate row to be written

w to be writte n						
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 session handle TPM Owner authentication
		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 session handle
10	20			TPM_AUTHDATA	ownerAuth	The authorization session digest. HMAC key:ownerAuth

3870Outgoing 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_LoadOwnerDelegation
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.

3871Actions

38721. Map blob to D1 a TPM_DELEGATE_OWNER_BLOB.

3873 a. Validate that D1 -> tag == TPM_TAG_DELEGATE_OWNER_BLOB

38742. Locate D1 -> pub -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate
3875 row, return TPM_BADINDEX if not found

38763. Set FR to TPM_FAMILY_TABLE -> famTableRow[familyRow]

38774. If TPM Owner is installed

3878 a. Validate the command and parameters using TPM Owner authentication, return
3879 TPM_AUTHFAIL on error

3880 b. If the command is delegated (authHandle session type is TPM_PID_DSAP or through
3881 ownerReference delegation), verify that D1 -> pub -> familyID matches authHandle ->
3882 familyID, on error return TPM_DELEGATE_FAMILY

38835. Else

3884 a. If tag is not TPM_TAG_RQU_COMMAND, return TPM_AUTHFAIL

3885 b. If FR -> flags -> TPM_DELEGATE_ADMIN_LOCK is TRUE return
3886 TPM_DELEGATE_LOCK

3887 c. Validate max NV writes without an owner

3888 i. Set NV1 to PD -> noOwnerNVWrite

3889 ii. Increment NV1 by 1

3890 iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITES

3891 iv. Set PD -> noOwnerNVWrite to NV1

38926. If FR -> flags -> TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD

38937. If TPM Owner is installed, validate the integrity of the blob

3894 a. Copy D1 -> integrityDigest to H2

3895 b. Set D1 -> integrityDigest to all zeros

3896 c. Create H3 the HMAC of D1 using tpmProof as the secret

3897 d. Compare H2 to H3, return TPM_AUTHFAIL on mismatch

38988. If TPM Owner is installed, create S1 a TPM_DELEGATE_SENSITIVE area by decrypting
3899 D1 -> sensitiveArea using TPM_DELEGATE_KEY. Otherwise set S1 = D1 -> sensitiveArea

39009. Validate S1

3901 a. Validate that S1 -> tag == TPM_TAG_DELEGATE_SENSITIVE, return
3902 TPM_INVALID_STRUCTURE on error

390310. Validate that index is a valid value for delegateTable, return TPM_BADINDEX on error

390411. The TPM invalidates sessions

3905 a. MUST invalidate all DSAP sessions

3906 b. MUST invalidate all OSAP sessions associated with the delegation table

3907 c. MAY invalidate any other session

390812. Copy data to the delegate table row

3909 a. Copy the TPM_DELEGATE_PUBLIC from D1 -> pub to TPM_DELEGATE_TABLE ->
3910 delRow[index] -> pub.

3911 b. Copy the TPM_SECRET from S1 -> authValue to TPM_DELEGATE_TABLE ->
3912 delRow[index] -> authValue.

3913 c. Set TPM_STCLEAR_DATA-> ownerReference to TPM_KH_OWNER

3914 d. If authHandle is of type DSAP then continueAuthSession MUST set to FALSE

391513. Return TPM_SUCCESS

19.5 TPM_Delegate_ReadTable

Start of informative comment:

This command reads 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 AuthData 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	1S	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	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Delegate_ReadTable
4	4	3S	4	UINT32	familyTableSize	Size in bytes of familyTable
5	<>	4S	<>	BYTE [ ]	familyTable	Array of TPM_FAMILY_TABLE_ENTRY elements
6	4	5S	4	UINT32	delegateTableSize	Size in bytes of delegateTable
7	<>	6S	<>	BYTE[]	delegateTable	Array of TPM_DELEGATE_INDEX and TPM_DELEGATE_PUBLIC elements

Actions

1. Set familyTableSize to the number of valid families on the TPM times sizeof(TPM_FAMILY_TABLE_ELEMENT).
2. Copy the valid entries in the internal family table to the output array familyTable
3. Set delegateTableSize to the number of valid delegate table entries on the TPM times (sizeof(TPM_DELEGATE_PUBLIC) + 4).
4. For each valid entry
   a. Write the TPM_DELEGATE_INDEX to delegateTable
   b. Copy the TPM_DELEGATE_PUBLIC to delegateTable

39365. Return TPM_SUCCESS

19.6 TPM_Delegate_UpdateVerification

Start of informative comment:

TPM_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_UpdateVerification
4	4	2S	4	UINT32	inputSize	The size of inputData
5	<>	3S	<>	BYTE	inputData	TPM_DELEGATE_KEY_BLOB or TPM_DELEGATE_OWNER_BLOB or TPM_DELEGATE_INDEX
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		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 session handle
9	20			TPM_AUTHDATA	ownerAuth	Authorization HMAC key: ownerAuth.

3944**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 session digest for the returned parameters. HMAC key: ownerAuth.

3945**Actions**

39461. Verify the TPM Owner, directly or indirectly through delegation, authorizes the command
3947 and parameters, on error return TPM_AUTHFAIL

39482. Determine the type of inputData (TPM_DELEGATE_TABLE_ROW or
3949 TPM_DELEGATE_OWNER_BLOB or TPM_DELEGATE_KEY_BLOB) and map D1 to that
3950 structure

3951 a. Mapping to TPM_DELEGATE_TABLE_ROW requires taking inputData as a tableIndex
3952 and locating the appropriate row in the table

39533. If D1 is a TPM_DELEGATE_OWNER_BLOB or TPM_DELEGATE_KEY_BLOB, validate the
3954 integrity of D1

3955 a. Copy D1 -> integrityDigest to H2

3956 b. Set D1 -> integrityDigest to all zeros

3957 c. Create H3 the HMAC of D1 using tpmProof as the secret

3958 d. Compare H2 to H3 return TPM_AUTHFAIL on mismatch

39594. Locate (D1 -> pub -> familyID) in the TPM_FAMILY_TABLE and set familyRow to indicate
3960 row, return TPM_BADINDEX if not found

39615. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]

39626. If delegated, verify that family of the delegated Owner-auth is the same as D1:
3963 (authHandle -> familyID) == (D1 -> pub -> familyID); otherwise return error
3964 TPM_DELEGATE_FAMILY

39657. If delegated, verify that the family of the delegated Owner-auth is enabled: if (authHandle
3966 -> familyID -> flags TPM_FAMFLAG_ENABLED) is FALSE, return TPM_DISABLED_CMD

39678. Set D1 -> verificationCount to FR -> verificationCount

9. If D1 is a TPM_DELEGATE_OWNER_BLOB or TPM_DELEGATE_KEY_BLOB set the integrity of D1
   a. Set D1 -> integrityDigest to all zeros
   b. Create H1 the HMAC of D1 using tpmProof as the secret
   c. Set D1 -> integrityDigest to H1
10. If D1 is a blob recreate the blob and return it

19.7 TPM_Delegate_VerifyDelegation

Start of informative comment:

TPM_VerifyDelegation interprets a delegate blob and returns success or failure, depending on whether the blob is currently valid. The delegate blob is NOT loaded 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_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_Delegate_VerifyDelegation
4	4	2S	4	UINT32	delegationSize	The length of the delegated information blob
5	<>	3S	<>	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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal, TPM_Delegate_VerifyDelegation

Actions

1. Determine the type of blob, If delegation -> tag is equal to TPM_TAG_DELGATE_OWNER_BLOB then
   a. Map D1 a TPM_DELEGATE_OWNER_BLOB to delegation
2. Else if delegation -> tag = TPM_TAG_DELG_KEY_BLOB
   a. Map D1 a TPM_DELEGATE_KEY_BLOB to delegation
3. Else return TPM_BAD_PARAMETER
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 -> pub -> verificationCount matches FR -> verificationCount, on mismatch return TPM_FAMILYCOUNT
8. Validate the integrity of D1
   a. Copy D1 -> integrityDigest to H2

3996 b. Set D1 -> integrityDigest to all zeros

3997 c. Create H3 the HMAC of D1 using tpmProof as the secret

3998 d. Compare H2 to H3 return TPM_AUTHFAIL on mismatch

39999. Create S1 a TPM_DELEGATE_SENSITIVE area by decrypting D1 -> sensitiveArea using 4000 TPM_DELEGATE_KEY

400110. Validate S1 values

4002 a. S1 -> tag is TPM_TAG_DELEGATE_SENSITIVE

4003 b. Return TPM_BAD_PARAMETER on error

400411. Return TPM_SUCCESS

4005 20. Non-volatile Storage

4006**Start of informative comment:**

4007This section handles the allocation and use of the TPM non-volatile storage.

4008**End of informative comment.**

4009If nvIndex refers to the DIR, the TPM ignores actions containing access control checks that
4010have no meaning for the DIR. The TPM only checks the owner authorization.

4012 20.1 TPM_NV_DefineSpace

4013 **Start of informative comment:**

4014 This establishes the space necessary for the indicated index. The definition will include the
4015 access requirements for writing and reading the area.

4016 Previously defined space at the index and new size is non-zero (and space is available,
4017 etc.) –> redefine the index

4018 No previous space at the index and new size is non-zero (and space is available,
4019 etc.)-> define the index

4020 Previously defined space at the index and new size is 0 -> delete the index

4021 No previous space at the index and new size is 0 -> error

4022 The space definition size does not include the area needed to manage the space.

4023 Setting TPM_PERMANENT_FLAGS -> nvLocked TRUE when it is already TRUE is not an
4024 error.

4025 **End of informative comment.**

4026 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	<>	2S	<>	TPM_NV_DATA_PUBLIC	pubInfo	The public parameters of the NV area
5	20	3S	20	TPM_ENCAUTH	encAuth	The encrypted AuthData, only valid if the attributes require subsequent authorization
6	4			TPM_AUTHHANDLE	authHandle	The authorization session 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 session handle
9	20			TPM_AUTHDATA	ownerAuth	The authorization session digest HMAC key: ownerAuth

4027 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, fixed to FALSE
6	20			TPM_AUTHDATA	ownerAuth	The authorization session digest HMAC key: ownerAuth

Description

For the case where pubInfo -> dataSize is 0, pubInfo -> pcrInfoRead and pubInfo -> pcrInfoWrite are not used. However, since the general principle is to validate parameters before changing state, the TPM SHOULD parse pubInfo completely before invalidating the data area.

Actions

1. If pubInfo -> nvIndex == TPM_NV_INDEX_LOCK and tag = TPM_TAG_RQU_COMMAND
   a. If pubInfo -> dataSize is not 0, the command MAY return TPM_BADINDEX.
   b. Set TPM_PERMANENT_FLAGS -> nvLocked to TRUE
   c. Return TPM_SUCCESS
2. If TPM_PERMANENT_FLAGS -> nvLocked is FALSE then all authorization checks except for the Max NV writes are ignored
   a. Ignored checks include physical presence, owner authorization, 'D' bit check, bGlobalLock, no authorization with a TPM owner present, bWriteSTClear, the check that pubInfo -> dataSize is 0 in Action 5.c. (the no-authorization case) , disabled and deactivated.
      i. The check that pubInfo -> dataSize is 0 is still enforced in Action 6.f. (returning after deleting a previously defined storage area) and Action 9.f. (not allowing a space of size 0 to be defined).
      ii. If ownerAuth is present, the TPM MAY check the authorization HMAC.
   b. The check for pubInfo -> nvIndex == TPM_NV_INDEX0 in Action 3. is not ignored.
3. If pubInfo -> nvIndex has the D bit (bit 28) set to a 1 or pubInfo -> nvIndex == TPM_NV_INDEX0 then
   a. Return TPM_BADINDEX
   b. The D bit specifies an index value that is set in manufacturing and can never be deleted or added to the TPM
   c. Index value TPM_NV_INDEX0 is reserved and cannot be defined
4. If tag = TPM_TAG_RQU_AUTH1_COMMAND then
   a. The TPM MUST validate the command and parameters using the TPM Owner authentication and ownerAuth, on error return TPM_AUTHFAIL
   b. authHandle session type MUST be OSAP
   c. Create A1 by decrypting encAuth according to the ADIP indicated by authHandle.
5. else
   a. Validate the assertion of physical presence. Return TPM_BAD_PRESENCE on error.

4062 b. If TPM Owner is present then return TPM_OWNER_SET.

4063 c. If pubInfo -> dataSize is 0 then return TPM_BAD_DATASIZE. Setting the size to 0
4064 represents an attempt to delete the value without TPM Owner authentication.

4065 d. Validate max NV writes without an owner

4066 i. Set NV1 to TPM_PERMANENT_DATA -> noOwnerNVWrite

4067 ii. Increment NV1 by 1

4068 iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITES

4069 iv. Set NV1_INCREMENTED to TRUE

4070 e. Set A1 to encAuth. There is no nonce or authorization to create the encryption string,
4071 hence the AuthData value is passed in the clear

40726. If pubInfo -> nvIndex points to a valid previously defined storage area then

4073 a. Map D1 a TPM_NV_DATA_SENSITIVE to the storage area

4074 b. If D1 -> attributes specifies TPM_NV_PER_GLOBALLOCK then

4075 i. If TPM_STCLEAR_FLAGS -> bGlobalLock is TRUE then return
4076 TPM_AREA_LOCKED

4077 c. If D1 -> attributes specifies TPM_NV_PER_WRITE_STCLEAR

4078 i. If D1 -> pubInfo -> bWriteSTClear is TRUE then return TPM_AREA_LOCKED

4079 d. Invalidate the data area currently pointed to by D1 and ensure that if the area is
4080 reallocated no residual information is left

4081 e. If NV1_INCREMENTED is TRUE

4082 i. Set TPM_PERMANENT_DATA -> noOwnerNVWrite to NV1

4083 f. The TPM invalidates authorization sessions

4084 i. MUST invalidate all authorization sessions associated with D1

4085 ii. MAY invalidate any other authorization session

4086 g. If pubInfo -> dataSize is 0 then return TPM_SUCCESS

40877. Parse pubInfo -> pcrInfoRead

4088 a. Validate pcrInfoRead structure on error return TPM_INVALID_STRUCTURE

4089 i. Validation includes proper PCR selections and locality selections

40908. Parse pubInfo -> pcrInfoWrite

4091 a. Validate pcrInfoWrite structure on error return TPM_INVALID_STRUCTURE

4092 i. Validation includes proper PCR selections and locality selections

4093 b. If pcrInfoWrite -> localityAtRelease disallows some localities

4094 i. Set writeLocalities to TRUE

4095 c. Else

4096 i. Set writeLocalities to FALSE

40979. Validate that the attributes are consistent

4098 a. The TPM SHALL ignore the bReadSTClear, bWriteSTClear and bWriteDefine
4099 attributes during the execution of this command

4100 b. If TPM_NV_PER_OWNERWRITE is TRUE and TPM_NV_PER_AUTHWRITE is TRUE
4101 return TPM_AUTH_CONFLICT

4102 c. If TPM_NV_PER_OWNERREAD is TRUE and TPM_NV_PER_AUTHREAD is TRUE
4103 return TPM_AUTH_CONFLICT

4104 d. If TPM_NV_PER_OWNERWRITE and TPM_NV_PER_AUTHWRITE and
4105 TPM_NV_PER_WRITEDEFINE and TPM_NV_PER_PPWRITE and writeLocalities are all
4106 FALSE

4107 i. Return TPM_PER_NOWRITE

4108 e. Validate pubInfo -> nvIndex

4109 i. Make sure that the index is applicable for this TPM. Return TPM_BADINDEX
4110 on error. A valid index is platform and context sensitive. That is, attempting to
4111 validate an index may be successful in one configuration and invalid in another
4112 configuration. The individual index values MUST indicate if there are any
4113 restrictions on the use of the index.

4114 ii. TPM_NV_INDEX_DIR is always an invalid defined index.

4115 f. If dataSize is 0 return TPM_BAD_PARAM_SIZE

411610. Create D1 a TPM_NV_DATA_SENSITIVE structure

4117 a. Set D1 -> pubInfo to pubInfo

4118 b. Set D1 -> authValue to A1

4119 c. Set D1 -> pubInfo -> bReadSTClear to FALSE

4120 d. Set D1 -> pubInfo -> bWriteSTClear to FALSE

4121 e. Set D1 -> pubInfo -> bWriteDefine to FALSE

412211. Validate that sufficient NV is available to store D1 and pubInfo -> dataSize bytes of data

4123 a. Return TPM_NOSPACE if pubInfo -> dataSize is not available in the TPM

412412. If pubInfo -> nvIndex is not TPM_NV_INDEX_TRIAL

4125 a. Reserve NV space for pubInfo -> dataSize

4126 b. Set all bytes in the newly defined area to 0xFF

4127 c. If NV1_INCREMENTED is TRUE

4128 i. Set TPM_PERMANENT_DATA -> noOwnerNVWrite to NV1

412913. Ignore continueAuthSession on input and set to FALSE on output

413014. 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.

The action setting bGlobalLock to TRUE is intentionally before the action checking the owner authorization. This allows code (e.g., a BIOS) to lock NVRAM without knowing the owner authorization.

The DIR (TPM_NV_INDEX_DIR) has the attributes TPM_NV_PER_OWNERWRITE and TPM_NV_WRITEALL.

Certain platform manufacturers or software might require specific error handling in Action 20.2.

Owner authorization is not required when nvLocked is FALSE. If the host does send owner authorization, Action 20.2 indicates that it should be correct, since some TPM implementations may validate it.

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 session 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 session handle
11	20			TPM_AUTHDATA	ownerAuth	The authorization session digest 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_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 session digest HMAC key: ownerAuth

4149Description

4150For TPM_NV_INDEX_DIR, the ordinal MUST NOT set an error code for the “if dataSize = 0”
4151action. However, the flags set in this case are not applicable to the DIR.

4152Actions

41531. If TPM_PERMANENT_FLAGS -> nvLocked is FALSE then all authorization checks except
4154 for the max NV writes are ignored

4155 a. Ignored checks include physical presence, owner authorization,
4156 TPM_NV_PER_OWNERWRITE, PCR, bWriteDefine, bGlobalLock, bWriteSTClear, locality,
4157 disabled and deactivated.

4158 b. TPM_NV_PER_AUTHWRITE is not ignored.

4159 c. If ownerAuth is present, the TPM MAY check the authorization HMAC.

41602. Locate and set D1 to the TPM_NV_DATA_AREA that corresponds to nvIndex, return
4161 TPM_BADINDEX on error

4162 a. If nvIndex = TPM_NV_INDEX_DIR, set D1 to TPM_PERMANENT_DATA -> authDir[0]

41633. If TPM_PERMANENT_FLAGS -> nvLocked is TRUE

4164 a. If D1 -> permission -> TPM_NV_PER_OWNERWRITE is TRUE

4165 i. If TPM_PERMANENT_FLAGS -> disable is TRUE, return TPM_DISABLED

4166 ii. If TPM_STCLEAR_FLAGS -> deactivated is TRUE, return TPM_DEACTIVATED

4167 b. If D1 -> permission -> TPM_NV_PER_OWNERWRITE is FALSE

4168 i. If TPM_PERMANENT_FLAGS -> disable is TRUE, the TPM MAY return
4169 TPM_DISABLED

4170 ii. If TPM_STCLEAR_FLAGS -> deactivated is TRUE, the TPM MAY return
4171 TPM_DEACTIVATED

41724. If tag = TPM_TAG_RQU_AUTH1_COMMAND then

4173 a. If D1 -> permission -> TPM_NV_PER_OWNERWRITE is FALSE return
4174 TPM_AUTH_CONFLICT

4175 i. This check is ignored if nvIndex is TPM_NV_INDEX0.

4176 b. Validate command and parameters using ownerAuth HMAC with TPM Owner
4177 authentication as the secret, return TPM_AUTHFAIL on error

41785. Else

4179 a. If D1 -> permission -> TPM_NV_PER_OWNERWRITE is TRUE return
4180 TPM_AUTH_CONFLICT

4181 i. This check is ignored if nvIndex is TPM_NV_INDEX0.

b. If no TPM Owner validate max NV writes without an owner
    i. Set NV1 to TPM_PERMANENT_DATA -> noOwnerNVWrite
    ii. Increment NV1 by 1
    iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITES
    iv. Set NV1_INCREMENTED to TRUE

6. If nvIndex is TPM_NV_INDEX0 then
    a. If dataSize is not 0, the TPM MAY return TPM_BADINDEX.
    b. Set TPM_STCLEAR_FLAGS -> bGlobalLock to TRUE
    c. Return TPM_SUCCESS

7. If D1 -> permission -> TPM_NV_PER_AUTHWRITE is TRUE return TPM_AUTH_CONFLICT

8. 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

9. If D1 -> attributes specifies TPM_NV_PER_PPWRITE then validate physical presence is asserted if not return TPM_BAD_PRESENCE

10. If D1 -> attributes specifies TPM_NV_PER_WRITEDEFINE
    a. If D1 -> bWriteDefine is TRUE return TPM_AREA_LOCKED

11. If D1 -> attributes specifies TPM_NV_PER_GLOBALLOCK
    a. If TPM_STCLEAR_FLAGS -> bGlobalLock is TRUE return TPM_AREA_LOCKED

12. If D1 -> attributes specifies TPM_NV_PER_WRITE_STCLEAR
    a. If D1 ->bWriteSTClear is TRUE return TPM_AREA_LOCKED

13. If D1 -> pcrInfoWrite -> pcrSelection specifies a selection of TPM_STCLEAR_DATA -> PCR[]
    a. Create P1 a composite hash of the TPM_STCLEAR_DATA -> PCR[] specified by D1 -> pcrInfoWrite
    b. Compare P1 to D1 -> pcrInfoWrite -> digestAtRelease return TPM_WRONGPCRVAL on mismatch

14. If dataSize = 0 then
    a. Set D1 -> bWriteSTClear to TRUE
    b. Set D1 -> bWriteDefine to TRUE

15. Else
    a. Set S1 to offset + dataSize
    b. If S1 > D1 -> dataSize return TPM_NOSPACE

c. If D1 -> attributes specifies TPM_NV_PER_WRITEALL
    i. If dataSize != D1 -> dataSize return TPM_NOT_FULLWRITE
d. Write the new value into the NV storage area
e. If NV1_INCREMENTED is TRUE
    i. Set TPM_PERMANENT_DATA -> noOwnerNVWrite to NV1
16. Set D1 -> bReadSTClear to FALSE
17. Return TPM_SUCCESS

4225 20.3 TPM_NV_WriteValueAuth

4226**Start of informative comment:**

4227This command writes to a previously defined area. The area must require authorization to 4228write. Use this command when authorization other than the owner authorization is to be 4229used. Otherwise, use TPM_NV_WriteValue.

4230The Part 2 ordinal table indicates that TPM_NV_WriteValueAuth requires an owner present. 4231This is normative, although it was a mistake.

4232**End of informative comment.**

4233 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 session 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 session handle
11	20			TPM_AUTHDATA	authValue	HMAC key: NV element auth value

4234 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

4235 Actions

42361. Locate and set D1 to the TPM_NV_DATA_AREA that corresponds to nvIndex, return 4237 TPM_BADINDEX 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 -> pcrInfoWrite -> pcrSelection specifies a selection of PCR
   a. Create P1 a composite hash of the TPM_STCLEAR_DATA -> PCR[] specified by D1 -> pcrInfoWrite
   b. Compare P1 to digestAtRelease return TPM_WRONGPCRVAL 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_NV_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 authentication.

Action 1 indicates that if the NV area is not locked then reading of the NV area continues without ANY authorization. This is intentional, and allows a platform manufacturer to set the NV areas, read them back, and then lock them all without having to install a TPM owner.

Certain platform manufacturers or software might require specific error handling in Action 20.4.

Owner authorization is not required when nvLocked is FALSE. If the host does send owner authorization, Action 20.4 indicates that it should be correct, since some TPM implementations may validate it.

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 session handle used for TPM Owner 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 caller
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	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		TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	TPM_ORD_NV_ReadValue
4	4	3S	4	UINT32	dataSize	The size of the data area
5	<>	4S	<>	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: ownerAuth

4284**Actions**

42851. If TPM_PERMANENT_FLAGS -> nvLocked is FALSE then all authorization checks are 4286 ignored.

4287 a. Ignored checks include physical presence, owner authorization, PCR, bReadSTClear, 4288 locality, TPM_NV_PER_OWNERREAD, disabled and deactivated.

4289 b. TPM_NV_PER_AUTHREAD is not ignored.

4290 c. If ownerAuth is present, the TPM MAY check the authorization HMAC.

42912. Set D1 a TPM_NV_DATA_AREA structure to the area pointed to by nvIndex, if not found 4292 return TPM_BADINDEX

4293 a. If nvIndex = TPM_NV_INDEX_DIR, set D1 to TPM_PERMANENT_DATA -> authDir[0]

42943. If TPM_PERMANENT_FLAGS -> nvLocked is TRUE

4295 a. If D1 -> permission -> TPM_NV_PER_OWNERREAD is TRUE

4296 i. If TPM_PERMANENT_FLAGS -> disable is TRUE, return TPM_DISABLED

4297 ii. If TPM_STCLEAR_FLAGS -> deactivated is TRUE, return TPM_DEACTIVATED

4298 b. If D1 -> permission -> TPM_NV_PER_OWNERREAD is FALSE

4299 i. If TPM_PERMANENT_FLAGS -> disable is TRUE, the TPM MAY return 4300 TPM_DISABLED

4301 ii. If TPM_STCLEAR_FLAGS -> deactivated is TRUE, the TPM MAY return 4302 TPM_DEACTIVATED

43034. If tag = TPM_TAG_RQU_AUTH1_COMMAND then

4304 a. If D1 -> TPM_NV_PER_OWNERREAD is FALSE return TPM_AUTH_CONFLICT

4305 b. Validate command and parameters using TPM Owners authentication on error return 4306 TPM_AUTHFAIL

43075. Else

4308 a. If D1 -> TPM_NV_PER_AUTHREAD is TRUE return TPM_AUTH_CONFLICT

4309 b. If D1 -> TPM_NV_PER_OWNERREAD is TRUE return TPM_AUTH_CONFLICT

43106. Check that D1 -> pcrInfoRead -> localityAtRelease for TPM_STANY_DATA -> 4311 localityModifier is TRUE

4312 a. For example if TPM_STANY_DATA -> localityModifier was 2 then D1 -> pcrInfo -> 4313 localityAtRelease -> TPM_LOC_TWO would have to be TRUE

4314 b. On error return TPM_BAD_LOCALITY

43157. If D1 -> attributes specifies TPM_NV_PER_PPREAD then validate physical presence is 4316 asserted if not return TPM_BAD_PRESENCE

43178. If D1 -> TPM_NV_PER_READ_STCLEAR then

a. If D1 -> bReadSTClear is TRUE return TPM_DISABLED_CMD

9. If D1 -> pcrInfoRead -> pcrSelection specifies a selection of PCR

a. Create P1 a composite hash of the TPM_STCLEAR_DATA -> PCR[] specified by D1 -> pcrInfoRead

b. Compare P1 to D1 -> pcrInfoRead -> digestAtRelease return TPM_WRONGPCRVAL on mismatch

10. If dataSize is 0 then

a. Set D1 -> bReadSTClear to TRUE

b. Set data to NULL (output parameter dataSize to 0)

11. Else

a. Set S1 to offset + dataSize

b. If S1 > D1 -> dataSize return TPM_NOSPACE

c. Set data to area pointed to by offset

i. This includes partial reads of TPM_NV_INDEX_DIR.

12. Return TPM_SUCCESS

4333 20.5 TPM_NV_ReadValueAuth

4334**Start of informative comment:**

4335This command requires that the read be authorized by a value set with the blob.

4336The Part 2 ordinal table indicates that TPM_NV_ReadValueAuth requires an owner present.
4337This is normative, although it was a mistake.

4338**End of informative comment.**

4339 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 session handle
10	20			TPM_AUTHDATA	authHmac	HMAC key: nv element authorization

4340 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

4341 Actions

43421. Locate and set D1 to the TPM_NV_DATA_AREA that corresponds to nvIndex, on error
4343 return TPM_BADINDEX

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 -> pcrInfoRead -> pcrSelection specifies a selection of PCR
   a. Create P1 a composite hash of the TPM_STCLEAR_DATA -> PCR[] specified by D1 -> pcrInfoRead
   b. Compare P1 to D1 -> pcrInfoRead -> digestAtRelease return TPM_WRONGPCRVAL 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 all zeros
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

Start of informative comment:

Three TPM_RT_CONTEXT session resources located in TPM_STANY_DATA work together to control session save and load: contextNonceSession, contextCount, and contextList[].

All three MUST initialized at TPM_Startup(ST_CLEAR) and TPM_Startup(ST_DEACTIVATED) and MAY be initialized at TPM_Startup(ST_STATE). Initializing invalidates all saved sessions. They MAY be restored by TPM_Startup(ST_STATE). This case would allow saved sessions to be loaded. The actual ST_STATE operation is reported by the TPM_RT_CONTEXT startup effect.

TPM_SaveContext creates a contextBlob containing an encrypted contextNonceSession. The nonce is checked by TPM_LoadContext. So initializing contextNonceSession invalidates all saved contexts. The nonce is large and protected, making a replay infeasible.

The contextBlob also contains a public but protected contextCount. The count increments for each saved contextBlob. The TPM also saves contextCount in contextList[]. The TPM validates contextBlob against the contextList[] during TPM_LoadContext. Since the contextList[] is finite, it limits the number of valid saved sessions. Since the contextCount cannot be allowed to wrap, it limits the total number of saved sessions.

After a contextBlob is loaded, its contextCount entry is removed from contextList[]. This releases space in the context list for future entries. It also invalidates the contextBlob. So a saved contextBlob can be loaded only once.

TPM_FlushSpecific can also specify a contextCount to be removed from the contextList[], allowing invalidation of an individual contextBlob. This is different from TPM_FlushSpecific specifying a session handle, which invalidates a loaded session, not a saved contextBlob.

End of informative comment.

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 non-volatile storage 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.

The key handle does not reference an authorized entity and is not validated.

The check for two remaining key slots ensures that users can load the two keys required to execute many commands. Since only the owner can flush owner evict keys, non-owner commands could be blocked if this test was not performed.

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	<>	2S	<>	TPM_PUBKEY	pubKey	The public key associated with the loaded key
6	4	3S	4	TPM_KEY_CONTROL	bitName	The name of the bit to be modified
7	1	4S	1	BOOL	bitValue	The value to set the bit to
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
9		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 session handle
12	20		20	TPM_AUTHDATA	ownerAuth	HMAC authorization: key ownerAuth

4406**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 ownerAuth

4407**Description**

44081. Set an internal bit within the key cache that controls some attribute of a loaded key.

4409**Actions**

44101. Validate the AuthData using the owner authentication value, on error return
4411 TPM_AUTHFAIL

44122. Validate that keyHandle refers to a loaded key, return TPM_INVALID_KEYHANDLE on
4413 error.

44143. Validate that pubKey matches the key held by the TPM pointed to by keyHandle, return
4415 TPM_BAD_PARAMETER on mismatch

4416 a. This check is added so that virtualization of the keyHandle does not result in attacks,
4417 as the keyHandle is not associated with an authorization value

44184. Validate that bitName is valid, return TPM_BAD_MODE on error.

44195. If bitName == TPM_KEY_CONTROL_OWNER_EVICT

4420 a. If bitValue == TRUE

4421 i. Verify that after this operation at least two key slots will be present within the
4422 TPM that can store any type of key both of which do NOT have the OwnerEvict bit
4423 set, on error return TPM_NOSPACE

4424 ii. Verify that for this key handle, parentPCRStatus is FALSE and isVolatile is
4425 FALSE. Return TPM_BAD_PARAMETER on error.

4426 iii. Set ownerEvict within the internal key storage structure to TRUE.

4427 b. Else if bitValue == FALSE

4428 i. Set ownerEvict within the internal key storage structure to FALSE.

44296. Return TPM_SUCCESS

21.2 TPM_SaveContext

Start of informative comment:

TPM_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.

There is no assumption that a saved context blob is stored in a safe, protected area. Since the context blob can be loaded at any time, do not rely on TPM_SaveContext to restrict access to an entity such as a key. If use of the entity should be restricted, means such as authorization secrets or PCR's should be used.

In general, TPM_SaveContext can save a transport session. However, it cannot save an exclusive transport session, because any ordinal other than TPM_ExecuteTransport terminates the exclusive transport session. This action prevents the exclusive transport session from being saved and reloaded while intervening commands are hidden from the transport log.

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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveContext
4	4			TPM_HANDLE	handle	Handle of the resource being saved.
5	4	2S	4	TPM_RESOURCE_TYPE	resourceType	The type of resource that is being saved
6	16	3S	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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveContext
4	4	3S	4	UINT32	contextSize	The actual size of the outgoing context blob
5	<>	4S	<>	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.

4451**Actions**

44521. Map V1 to TPM_STANY_DATA

44532. Validate that handle points to resource that matches resourceType, return
4454 TPM_INVALID_RESOURCE on error

44553. Validate that resourceType is a resource from the following list if not return
4456 TPM_INVALID_RESOURCE

4457 a. TPM_RT_KEY

4458 b. TPM_RT_AUTH

4459 c. TPM_RT_TRANS

4460 d. TPM_RT_DAA_TPM

44614. Locate the correct nonce

4462 a. If resourceType is TPM_RT_KEY

4463 i. If TPM_STCLEAR_DATA -> contextNonceKey is all zeros

4464 (1) Set TPM_STCLEAR_DATA -> contextNonceKey to the next value from
4465 the TPM RNG

4466 ii. Map N1 to TPM_STCLEAR_DATA -> contextNonceKey

4467 iii. If the key has TPM_KEY_CONTROL_OWNER_EVICT set then return
4468 TPM_OWNER_CONTROL

4469 b. Else

4470 i. If V1 -> contextNonceSession is all zeros

4471 (1) Set V1 -> contextNonceSession to the next value from the TPM RNG

4472 ii. Map N1 to V1 -> contextNonceSession

44735. Set K1 to TPM_PERMANENT_DATA -> contextKey

44746. Create R1 by putting the sensitive part of the resource pointed to by handle into a
4475 structure. The structure is a TPM manufacturer option. The TPM MUST ensure that ALL
4476 sensitive information of the resource is included in R1.

44777. Create C1 a TPM_CONTEXT_SENSITIVE structure

4478 a. C1 forms the inner encrypted wrapper for the blob. All saved context blobs MUST
4479 include a TPM_CONTEXT_SENSITIVE structure and the TPM_CONTEXT_SENSITIVE
4480 structure MUST be encrypted.

4481 b. Set C1 -> contextNonce to N1

4482 c. Set C1 -> internalData to R1

44838. Create B1 a TPM_CONTEXT_BLOB

4484 a. Set B1 -> tag to TPM_TAG_CONTEXTBLOB

4485 b. Set B1 -> resourceType to resourceType

4486 c. Set B1 -> handle to handle

4487 d. Set B1 -> integrityDigest to all zeros

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).

i. For OSAP sessions and for DSAP sessions attached to keys, the hash of the entity MUST be included in additionalData

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. 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

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

iii. Increment V1 -> contextCount by 1

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 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

21.3 TPM_LoadContext

Start of informative comment:

TPM_LoadContext loads into the TPM a previously saved context. The command returns 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadContext
4	4			TPM_HANDLE	entityHandle	The handle the TPM MUST use to locate the entity tied to the OSAP/DSAP session
5	1	2S	1	BOOL	keepHandle	Indication if the handle MUST be preserved
6	4	3S	4	UINT32	contextSize	The size of the following context blob.
7	<>	4S	<>	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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadContext
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 -> contextNonce 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 and DSAP connected to the key) return TPM_RESOURCEMISSING

(1) For OSAP sessions and for DSAP sessions attached to keys, the TPM MUST validate that the hash of the entity matches the entity held by the TPM

(2) For OSAP and DSAP sessions referring to a key, verify that entityHandle identifies the key linked to this OSAP/DSAP session, if not return TPM_BAD_HANDLE.

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 TPM_STCLEAR_DATA -> contextNonceKey return TPM_BADCONTEXT

5. Validate the structure

a. Set H1 to B1 -> integrityDigest

b. Set B1 -> integrityDigest to all zeros

c. Copy M1 to B1 -> sensitiveData

d. Create H2 the HMAC of B1 using TPM_PERMANENT_DATA -> tpmProof as the HMAC key

e. If H2 does not 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_BAD_HANDLE

8. Else

a. The TPM SHOULD attempt to restore the handle but if not possible it MAY set the handle to any valid for B1 -> resourceType

9. If B1 -> resourceType is NOT TPM_RT_KEY

a. Find contextIndex such that V1 -> contextList[contextIndex] equals B1 -> TPM_CONTEXT_BLOB -> contextCount

b. If not found then return TPM_BADCONTEXT

c. Set V1 -> contextList[contextIndex] to 0

10. Process B1 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_EvictKey
- TPM_Reset

4587 22.1 TPM_FlushSpecific

4588**Start of informative comment:**

4589TPM_FlushSpecific flushes from the TPM a specific handle.

4590**End of informative comment.**

4591 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_FlushSpecific
4	4			TPM_HANDLE	handle	The handle of the item to flush
5	4	2S	4	TPM_RESOURCE_TYPE	resourceType	The type of resource that is being flushed

4592 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	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_FlushSpecific

4593 Description

4594TPM_FlushSpecific releases the resources associated with the given handle.

4595 Actions

45961. If resourceType is TPM_RT_CONTEXT

4597 a. The handle for a context is not a handle but the "context count" value. The TPM uses
4598 the "context count" value to locate the proper contextList entry and sets R1 to the
4599 contextList entry

46002. Else if resourceType is TPM_RT_KEY

4601 a. Set R1 to the key pointed to by handle

4602 b. If R1 -> ownerEvict is TRUE return TPM_KEY_OWNER_CONTROL

46033. Else if resourceType is TPM_RT_AUTH

4604 a. Set R1 to the authorization session pointed to by handle

46054. Else if resourceType is TPM_RT_TRANS

4606 a. Set R1 to the transport session pointed to by handle

46075. Else if resourceType is TPM_RT_DAA_TPM

4608 a. Set R1 to the DAA session pointed to by handle

46096. Else return TPM_INVALID_RESOURCE

46107. Validate that R1 determined by resourceType and handle points to a valid allocated 4611 resource. Return TPM_BAD_PARAMETER on error.

46128. Invalidate R1 and all internal resources allocated to R1

4613 a. Resources include authorization sessions

4614 23. Timing Ticks

4615**Start of informative comment:**

4616The TPM timing ticks are always available for use. The association of timing ticks to actual
4617time is a protocol that occurs outside of the TPM. See the design document for details.

4618The setting of the clock type variable is a one time operation that allows the TPM to be
4619configured to the type of platform that is installed on.

4620The ability for the TPM to continue to increment the timer ticks across power cycles of the
4621platform is a TPM and platform manufacturer decision.

4622**End of informative comment.**

4623 23.1 TPM_GetTicks

4624**Start of informative comment:**

4625This command returns the current tick count of the TPM.

4626**End of informative comment.**

4627 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	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_GetTicks

4628 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_GetTicks
4	32	3S	32	TPM_CURRENT_TICKS	currentTime	The current time held in the TPM

4629 Description

4630This command returns the current time held in the TPM. It is the responsibility of the
4631external system to maintain any relation between this time and a UTC value or local real
4632time value.

4633 Actions

46341. Set T1 to the internal TPM_CURRENT_TICKS structure

46352. Return T1 as currentTime.

23.2 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 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 session 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 session handle
10	20			TPM_AUTHDATA	privAuth	The authorization session digest that authorizes the use of keyHandle. HMAC key: key.usageAuth

4642Outgoing 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	32	3S	32	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 session digest for the returned parameters. HMAC key: key.usageAuth

4643Description

4644The function performs a digital signature on the hash of digestToStamp and the current tick
4645count.

4646It is the responsibility of the external system to maintain any relation between tick count
4647and a UTC value or local real time value.

4648Actions

46491. The TPM validates the AuthData to use the key pointed to by keyHandle.

46502. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING, TPM_KEY_IDENTITY or
4651 TPM_KEY_LEGACY, if not return the error code TPM_INVALID_KEYUSAGE.

46523. Validate that keyHandle -> sigScheme is TPM_SS_RSASSAPKCS1v15_SHA1 or
4653 TPM_SS_RSASSAPKCS1v15_INFO, if not return TPM_INAPPROPRIATE_SIG.

46544. If TPM_STCLEAR_DATA -> currentTicks is not properly initialized

4655 a. Initialize the TPM_STCLEAR_DATA -> currentTicks

46565. Create T1, a TPM_CURRENT_TICKS structure.

46576. Create H1 a TPM_SIGN_INFO structure and set the structure defaults

4658 a. Set H1 -> fixed to “TSTP”

4659 b. Set H1 -> replay to antiReplay

4660 c. Create H2 the concatenation of digestToStamp || T1

4661 d. Set H1 -> dataLen to the length of H2

4662 e. Set H1 -> data to H2

46637. The TPM computes the signature, sig, using the key referenced by keyHandle, using
4664 SHA-1 of H1 as the information to be signed

46658. The TPM returns T1 as currentTicks parameter

24. Transport Sessions

See Part 1 for rationale and security issues.

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.

The only restriction on what can happen inside of a transport session is that there is no "nesting" of sessions. It is permissible to perform operations that delete internal state and make the TPM inoperable.

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 session
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 session 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 session handle
11	20			TPM_AUTHDATA	keyAuth	Authorization. HMAC key: encKey.usageAuth

4680**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			TPM_TRANSHANDLE	transHandle	The handle for the transport session
5	4	3S	4	TPM_MODIFIER_INDICATOR	locality	The locality that called this command
6	32	4S	32	TPM_CURRENT_TICKS	currentTicks	The current tick count
7	20	5S	20	TPM_NONCE	transNonceEven	The even nonce in use for subsequent execute transport
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	Authorization. HMAC key: key.usageAuth

4681**Description**

4682This command establishes the transport sessions shared secret. The encryption of the
4683shared secret uses the public key of the key loaded in encKey.

4684**Actions**

46851. If encHandle is TPM_KH_TRANSPORT then

4686 a. If tag is NOT TPM_TAG_RQU_COMMAND return TPM_BADTAG

4687 b. If transPublic -> transAttributes specifies TPM_TRANSPORT_ENCRYPT return
4688 TPM_BAD_SCHEME

4689 c. If secretSize is not 20 return TPM_BAD_PARAM_SIZE

4690 d. Set A1 to secret

46912. Else

4692 a. encHandle -> keyUsage MUST be TPM_KEY_STORAGE or TPM_KEY_LEGACY return
4693 TPM_INVALID_KEYUSAGE on error

4694 b. If encHandle -> authDataUsage does not equal TPM_AUTH_NEVER and tag is NOT
4695 TPM_TAG_RQU_AUTH1_COMMAND return TPM_AUTHFAIL

4696 c. Using encHandle -> usageAuth validate the AuthData to use the key and the
4697 parameters to the command

4698 d. Create K1 a TPM_TRANSPORT_AUTH structure by decrypting secret using the key
4699 pointed to by encHandle

4700 e. Validate K1 for tag

4701 f. Set A1 to K1 -> authData

23. If transPublic -> transAttributes has TPM_TRANSPORT_ENCRYPT
    a. If TPM_PERMANENT_FLAGS -> FIPS is true and transPublic -> algId is equal to TPM_ALG_MGF1 return TPM_INAPPROPRIATE_ENC
    b. Check if the transPublic -> algId is supported, if not return TPM_BAD_KEY_PROPERTY
    c. If transPublic -> algid is TPM_ALG_AESXXX, check that transPublic -> encScheme is supported, if not return TPM_INAPPROPRIATE_ENC
    d. Perform any initializations necessary for the algorithm
24. Generate transNonceEven from the TPM RNG
25. Create T1 a TPM_TRANSPORT_INTERNAL structure
    a. Ensure that the TPM has sufficient internal space to allocate the transport session, return TPM_RESOURCES on error
    b. Assign a T1 -> transHandle value. This value is assigned by the TPM
    c. Set T1 -> transDigest to all zeros
    d. Set T1 -> transPublic to transPublic
    e. Set T1-> transNonceEven to transNonceEven
    f. Set T1 -> authData to A1
26. If TPM_STANY_DATA -> currentTicks is not properly initialized
    a. Initialize the TPM_STANY_DATA -> currentTicks
27. Set currentTicks to TPM_STANY_DATA -> currentTicks
28. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_LOG set then
    a. Create L1 a TPM_TRANSPORT_LOG_IN structure
        i. Set L1 -> parameters to SHA-1 (ordinal || transPublic || secretSize || secret)
        ii. Set L1 -> pubKeyHash to all zeros
        iii. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || L1)
    b. Create L2 a TPM_TRANSPORT_LOG_OUT structure
        i. Set L2 -> parameters to SHA-1 (returnCode || ordinal || locality || currentTicks || transNonceEven)
        ii. Set L2 -> locality to the locality of this command
        iii. Set L2 -> currentTicks to currentTicks, this MUST be the same value that is returned in the currentTicks parameter
        iv. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || L2)
29. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_EXCLUSIVE then set TPM_STANY_FLAGS -> transportExclusive to TRUE
    a. Execution of any command other than TPM_ExecuteTransport or TPM_ReleaseTransportSigned targeting this transport session will cause the abnormal invalidation of this transport session transHandle

b. The TPM gives no indication, other than invalidation of transHandle, that the session is terminated

10. 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.

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

The only restriction on what can happen inside of a transport session is that there is no "nesting" of sessions. It is permissible to perform operations that delete internal state and make the TPM inoperable.

Because, in general, key handles are not logged, a digest of the corresponding public key is logged. In cases where the key handle is logged (e.g. TPM_OwnerReadInternalPub), the public key is also logged.

The wrapped command is audited twice – once according to the actions of TPM_ExecuteTransport and once within the wrapped command itself according to the special rules for auditing a command wrapped in an encrypted transport session.

The method of incrementing the symmetric key counter value is different from that used by some standard crypto libraries (e.g. openSSL, Java JCE) that increment the entire counter value. TPM users should be aware of this to avoid errors when the counter wraps.

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	continueTransSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	transAuth	HMAC for transHandle key: transHandle -> authData

4764**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	TPM_MODIFIER_INDICATOR	locality	The locality that called this command
6	4	5S	4	UINT32	wrappedRspSize	Size of the wrapped response
7	<>	6S	<>	BYTE[]	wrappedRsp	The wrapped response
8	20	2H1	20	TPM_NONCE	transNonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	transNonceOdd	Nonce generated by caller
9	1	4H1	1	BOOL	continueTransSession	The continue use flag for the session
10	20			TPM_AUTHDATA	transAuth	HMAC for transHandle key: transHandle -> authData

4765**Description**

47661. This command executes a TPM command using the transport session.

47672. Prior to execution of the wrapped command (action 11 below) failure of the transport
4768 session MUST have no effect on the resources referenced by the wrapped command. The
4769 exception is when the TPM goes into failure mode and return FAILED_SELFTEST for all
4770 subsequent commands.

47713. After execution of the wrapped command, failure of the transport session MAY NOT
4772 affect wrapped command resources. That is, the TPM is not required to clean up the
4773 effects of the wrapped command. Sessions and keys MAY remain loaded. It is
4774 understood that the transport session will be returning an error code and not reporting
4775 any session nonces. Therefore, wrapped sessions are no longer useful to the caller. It is
4776 the responsibility of the caller to clean up the result of the wrapped command.

47774. Execution of the wrapped command (action 11) SHOULD have no effect on the transport
4778 session.

4779 a. The wrapped command SHALL use no resources of the transport session, this
4780 includes authorization sessions

4781 b. If the wrapped command execution returns an error (action 11 below) then the
4782 sessions for TPM_ExecuteTransport still operate properly.

4783 c. The exception to this is when the wrapped command causes the TPM to go into
4784 failure mode and return TPM_FAILSELFTEST for all subsequent commands

47855. Field layout

4786 a. Notation

4787i. et indicates the outer TPM_**E**xecute**T**ransport command and response

ii. w indicates the inner command and response that is **w**rapped by the TPM_ExecuteTransport.

iii. (o) indicates **o**ptional parameters that may or may not be present in the wrapped command.

b. Command representation

c. **********

d. TAGet | LENet | ORDet | wrappedCmdSize | wrappedCmd | AUTHet

e. **********

f. wrappedCmd looks like the following

g. **********

h. TAGw | LENw | ORDw | HANDLESw(o) | DATAw | AUTH1w (o) | AUTH2w (o)

i. **********

j. | LEN1 |

k. | E1 | (encrypted)

l. | C1 | (decrypted)

m. Response representation

n. **********

o. TAGet | LENet | RCet | … | wrappedRspSize | wrappedRsp | AUTHet

p. **********

q. wrappedRsp looks like the following

r. **********

s. TAGw | LENw | RCw | HANDLESw(o) | DATAw | AUTH1w (o) | AUTH2w (o)

t. **********

u. | LEN2 |

v. | ←---------- C2 ----------→ |

w. | S2 | (decrypted)

x. | E2 | (encrypted)

y. The only command and response parameter that is possibly encrypted is DATAw.

6. Additional DATAw comments

a. For TPM_FlushSpecific and TPM_SaveContext

i. The DATAw part of these commands does not include the handle.

(1) It is understood that encrypting the resourceType prevents a determination of the handle type.

ii. If the resourceType is TPM_RT_KEY, then the public key MUST be logged.

b. For TPM_DAA_Join and TPM_DAA_Sign

i. The DATAw part of these commands does not include the input handle. The output handle from stage 0 is included in DATAW.

c. For TPM_LoadKey2

i. The outgoing handle is not part of the outgoing DATAw and is not encrypted or logged by the outgoing transport.

d. For TPM_LoadKey

i. The outgoing handle is part of the outgoing DATAw and is encrypted and logged.

e. For TPM_LoadContext

i. The outgoing handle is not part of the outgoing DATAw and is not encrypted or logged by the outgoing transport.

(1) It is understood that encrypting the contextBlob prevents a determination of the handle type.

f. For TPM_OIAP and TPM_OSAP, no input or output parameters are encrypted or logged.

i. For TPM_OSAP, the public key MUST NOT be logged. During a logged transport session, when a wrapped command uses the key, the public key referenced by the key handle will be logged in the transport. Thus, the audit trail is established for any key usage at that time.

g. For TPM_DSAP

i. For input, only entityValue is encrypted and logged.

ii. For output no parameters are encrypted or logged.

7. TPM_ExecuteTransport returns an implementation defined result when the wrapped command would cause termination of the transport session. Implementation defined possibilities include but are not limited to: the wrapped command may execute, completely, partially, or not at all, the transport session may or may not be terminated, continueTransSession may not be processed or returned correctly, and an error may or may not be returned. The wrapped commands include:

a. TPM_FlushSpecific, TPM_SaveContext targeting the transport session

b. TPM_OwnerClear, TPM_ForceClear, TPM_RevokeTrust

Actions

1. Using transHandle locate the TPM_TRANSPORT_INTERNAL structure T1

2. Parse wrappedCmd

a. Set TAGw, LENw, and ORDw to the parameters from wrappedCmd

b. Set E1 to DATAw

i. This pointer is ordinal dependent and requires the execute transport command to parse wrappedCmd

c. Set LEN1 to the length of DATAw

i. DATAw always ends at the start of AUTH1w if AUTH1w is present

3. If LEN1 is less than 0, or if ORDw is unknown, unimplemented, or cannot be determined
   a. Return TPM_BAD_PARAMETER
4. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_ENCRYPT set then
   a. If T1 -> transPublic -> algId is TPM_ALG_MGF1
      i. Using the MGF1 function, create string G1 of length LEN1. 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 E1.
   b. If the encryption algorithm requires an IV or CTR, calculate the IV or CTR value
      i. Using the MGF1 function, create string IV1 or CTR1 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. Note that any terminating characters within the string “in” are ignored, so a total of 42 bytes are hashed.
      ii. The symmetric key is taken from the first bytes of T1 -> authData.
      iii. Decrypt DATAw and replace the DATAw area of E1 creating C1
   c. TPM_OSAP, TPM_OIAP have no parameters encrypted
   d. TPM_DSAP has special rules for parameter encryption
5. Else
   a. Set C1 to the DATAw area E1 of wrappedCmd
6. Create H1 the SHA-1 of (ORDw || C1).
   a. C1 MUST point at the decrypted DATAw area of E1
   b. The TPM MAY use this calculation for both execute transport authorization, authorization of the wrapped command and transport log creation
7. Validate the incoming transport session authorization
   a. Set inParamDigest to SHA-1 (ORDet || wrappedCmdSize || H1)
   b. Calculate the HMAC of (inParamDigest || transLastNonceEven || transNonceOdd || continueTransSession) using T1 -> authData as the HMAC key
   c. Validate transAuth, on errors return TPM_AUTHFAIL
8. If TPM_ExecuteTransport requires auditing
   a. Create TPM_AUDIT_EVENT_IN using H1 as the input parameter digest and update auditDigest
   b. On any error return TPM_AUDITFAIL_UNSUCCESSFUL
9. If ORDw is from the list of following commands return TPM_NO_WRAP_TRANSPORT
   a. TPM_EstablishTransport

b. TPM_ExecuteTransport

c. TPM_ReleaseTransportSigned

10. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_LOG set then

a. Create L2 a TPM_TRANSPORT_LOG_IN structure

b. Set L2 -> parameters to H1

c. If ORDw is a command with no key handles

i. Set L2 -> pubKeyHash to all zeros

d. If ORDw is a command with one key handle

i. Create K2 the hash of the TPM_STORE_PUBKEY structure of the key pointed to by the key handle.

ii. Set L2 -> pubKeyHash to SHA-1 (K2)

e. If ORDw is a command with two key handles

i. Create K2 the hash of the TPM_STORE_PUBKEY structure of the key pointed to by the first key handle.

ii. Create K3 the hash of the TPM_STORE_PUBKEY structure of the key pointed to by the second key handle.

iii. Set L2 -> pubKeyHash to SHA-1 (K2 || K3)

f. Set T1 -> transDigest to the SHA-1 (T1 -> transDigest || L2)

g. If ORDw is a command with key handles, and the key is not loaded, return TPM_INVALID_KEYHANDLE.

11. Send the wrapped command to the normal TPM command parser, the output is C2 and the return code is RCw

a. If ORDw is a command that is audited then the TPM MUST perform the input and output audit of the command as part of this action.

b. The TPM MAY use H1 as the data value in the authorization and audit calculations during the execution of C1

12. Set CT1 to TPM_STANY_DATA -> currentTicks -> currentTicks and return CT1 in the currentTicks output parameter

13. Calculate S2 the pointer to the DATAw area of C2

a. Calculate LEN2 the length of S2 according to the same rules that calculated LEN1

14. Create H2 the SHA-1 of (RCw || ORDw || S2)

a. The TPM MAY use this calculation for execute transport authorization and transport log out creation

15. Calculate the outgoing transport session authorization

a. Create the new transNonceEven for the output of the command

b. Set outParamDigest to SHA-1 (RCet || ORDet || TPM_STANY_DATA -> currentTicks -> currentTicks || locality || wrappedRspSize || H2)

c. Calculate transAuth, the HMAC of (outParamDigest || transNonceEven || transNonceOdd || continueTransSession) using T1 -> authData as the HMAC key

16. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_LOG set then

a. Create L3 a TPM_TRANSPORT_LOG_OUT structure

b. Set L3 -> parameters to H2

c. Set L3 -> currentTicks to TPM_STANY_DATA -> currentTicks

d. Set L3 -> locality to TPM_STANY_DATA -> localityModifier

e. Set T1 -> transDigest to the SHA-1 (T1 -> transDigest || L3)

17. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_ENCRYPT set then

a. If T1 -> transPublic -> algId is TPM_ALG_MGF1

i. Using the MGF1 function, create string G2 of length LEN2. 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 E2 by performing an XOR of G2 and C2 starting at S2.

b. Else

i. Create IV2 or CTR2 using the same algorithm as IV1 or CTR1 with the input values transNonceEven, transNonceOdd, and “out”. Note that any terminating characters within the string “out” are ignored, so a total of 43 bytes are hashed.

ii. The symmetric key is taken from the first bytes of T1 -> authData

iii. Create E2 by encrypting C2 starting at S2

18. Else

a. Set E2 to the DATAw area S2 of wrappedRsp

19. If continueTransSession 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 or TPM_AUDITFAIL_UNSUCCESSFUL depending on RCw

21. Return C2 but with S2 replaced by E2 in the wrappedRsp 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 digital signature of the hash of all operations performed during the session.

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

This command uses two authorization sessions, 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	keyHandle	Handle of a loaded 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 session handle
9	20			TPM_AUTHDATA	keyAuth	The authorization session digest that authorizes the use of key. HMAC key: key -> usageAuth
10	4			TPM_TRANSHANDLE	transHandle	The transport session handle
		2H2	20	TPM_NONCE	transLastNonceEven	Even nonce in use by execute Transport
11	20	3H2	20	TPM_NONCE	transNonceOdd	Nonce supplied by caller for transport session
12	1	4H2	1	BOOL	continueTransSession	The continue use flag for the authorization session handle
13	20			TPM_AUTHDATA	transAuth	HMAC for transport session key: transHandle -> authData

4981Outgoing 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	4	3S	4	TPM_MODIFIER_INDICATOR	locality	The locality that called this command
5	32	4S	32	TPM_CURRENT_TICKS	currentTicks	The current ticks when the command executed
6	4	5S	4	UINT32	signSize	The size of the signature area
7	<>	6S	<>	BYTE[]	signature	The signature of the digest
8	20	2H1	20	TPM_NONCE	authNonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	authNonceOdd	Nonce generated by caller
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the session
10	20			TPM_AUTHDATA	keyAuth	HMAC: key -> usageAuth
11	20	2H2	20	TPM_NONCE	transNonceEven	Even nonce newly generated by TPM
		3H2	20	TPM_NONCE	transNonceOdd	Nonce generated by caller
12	1	4H2	1	BOOL	continueTransSession	The continue use flag for the session
13	20			TPM_AUTHDATA	transAuth	HMAC: transHandle -> authData

4982Description

4983This command releases a transport session and signs the transport log

4984Actions

49851. Using transHandle locate the TPM_TRANSPORT_INTERNAL structure T1

49862. Validate that keyHandle -> sigScheme is TPM_SS_RSASSAPKCS1v15_SHA1 or
4987 TPM_SS_RSASSAPKCS1v15_INFO, if not return TPM_INAPPROPRIATE_SIG.

49883. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING, if not return
4989 TPM_INVALID_KEYUSAGE

49904. Using key -> authData validate the command and parameters, on error return
4991 TPM_AUTHFAIL

49925. Using transHandle -> authData validate the command and parameters, on error return
4993 TPM_AUTH2FAIL

49946. If T1 -> transAttributes has TPM_TRANSPORT_LOG set then

4995 a. Create A1 a TPM_TRANSPORT_LOG_OUT structure

4996 b. Set A1 –> parameters to the SHA-1 (ordinal || antiReplay)

4997 c. Set A1 -> currentTicks to TPM_STANY_DATA -> currentTicks

4998 d. Set A1 -> locality to the locality modifier for this command

e. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || A1)

7. Else

a. Return TPM_BAD_MODE

8. 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 SHA-1 hash of H1 using the key pointed to by keyHandle

9. Invalidate all session data related to T1

10. Set continueTransSession to FALSE

11. Return TPM_SUCCESS

5010 25. Monotonic Counter

5011 25.1 TPM_CreateCounter

5012 **Start of informative comment:**

5013 This command creates the counter but does not select the counter. Counter creation
5014 assigns an AuthData value to the counter and sets the counters original start value. The
5015 original start value is the current internal base value plus one. Setting the new counter to
5016 the internal base avoids attacks on the system that are attempting to use old counter
5017 values.

5018 **End of informative comment.**

5019 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	encAuth	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 session handle used for owner authentication.
		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	Ignored
10	20		20	TPM_AUTHDATA	ownerAuth	Authorization ownerAuth.

5020 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	Fixed value of FALSE
8	20		20	TPM_AUTHDATA	resAuth	Authorization. HMAC key: ownerAuth.

5021 Description

5022This command creates a new monotonic counter. The TPM MUST support a minimum of 4
5023concurrent counters.

5024Actions

5025The TPM SHALL do the following:

50261. Using the authHandle field, validate the owner's AuthData to execute the command and
5027 all of the incoming parameters. The authorization session MUST be OSAP or DSAP

50282. Ignore continueAuthSession on input and set continueAuthSession to FALSE on output

50293. Create a1 by decrypting encAuth according to the ADIP indicated by authHandle.

50304. Validate that there is sufficient internal space in the TPM to create a new counter. If
5031 there is insufficient space, the command returns an error.

5032 a. The TPM MUST provide storage for a1, TPM_COUNTER_VALUE, countID, and any
5033 other internal data the TPM needs to associate with the counter

50345. Increment the max counter value

50356. Set the counter to the max counter value

50367. Set the counter label to label

50378. Create a countID

5038 25.2 TPM_IncrementCounter

5039**Start of informative comment:**

5040This authorized command increments the indicated counter by one. Once a counter has 5041been incremented then all subsequent increments must be for the same handle until a 5042successful TPM_Startup(ST_CLEAR) is executed.

5043The order for checking validation of the command parameters when no counter is active, 5044keeps an attacker from creating a denial-of-service attack.

5045**End of informative comment.**

5046 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 session 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 session handle
8	20			TPM_AUTHDATA	counterAuth	The authorization session digest that authorizes the use of countID. HMAC key: countID -> authData

5047 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 session digest for the returned parameters. HMAC key: countID -> authData

5048 Description

5049This function increments the counter by 1.

5050The TPM MAY implement increment throttling to avoid burn problems

5051**Actions**

50521. If TPM_STCLEAR_DATA -> countID is 0
5053 a. Validate that countID is a valid counter, return TPM_BAD_COUNTER on mismatch
5054 b. Validate the command parameters using counterAuth
5055 c. Set TPM_STCLEAR_DATA -> countID to countID
50562. else
5057 a. If TPM_STCLEAR_DATA -> countID does not equal countID
5058 i. Return TPM_BAD_COUNTER
5059 b. Validate the command parameters using counterAuth
50603. Increments the counter by 1
50614. Return new count value in count

5062 25.3 TPM_ReadCounter

5063**Start of informative comment:**

5064Reading the counter provides the caller with the current number in the sequence.

5065**End of informative comment.**

5066 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadCounter
4	4	2S	4	TPM_COUNT_ID	countID	ID value of the counter

5067 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReadCounter
4	10	3S	10	TPM_COUNTER_VALUE	count	The counter value

5068 Description

5069This returns the current value for the counter indicated. The counter MAY be any valid
5070counter.

5071 Actions

50721. Validate that countID points to a valid counter. Return TPM_BAD_COUNTER on error.

50732. Return count

5074 25.4 TPM_ReleaseCounter

5075**Start of informative comment:**

5076This command releases a counter such that no reads or increments of the indicated counter
5077will succeed.

5078**End of informative comment.**

5079 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 session 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 session handle
8	20			TPM_AUTHDATA	counterAuth	The authorization session digest that authorizes the use of countID. HMAC key: countID -> authData

5080 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 session digest for the returned parameters. HMAC key: countID -> authData

5081 Actions

5082The TPM uses countID to locate a valid counter.

50831. Authenticate the command and the parameters using the AuthData pointed to by
5084 countID. Return TPM_AUTHFAIL on error

50852. The TPM invalidates all internal information regarding the counter. This includes
5086 releasing countID such that any subsequent attempts to use countID will fail.

50873. The TPM invalidates sessions

a. MUST invalidate all OSAP sessions associated with the counter

b. MAY invalidate any other session

4. If TPM_STCLEAR_DATA -> countID equals countID,

a. Set TPM_STCLEAR_DATA -> countID to an illegal value (not the zero value)

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 session handle used for owner authentication
		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 session handle
8	20			TPM_AUTHDATA	ownerAuth	The authorization session digest that authorizes the inputs. 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_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 session digest for the returned parameters. HMAC key: ownerAuth

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.

51053. The TPM invalidates all internal information regarding the counter. This includes 5106 releasing countID such that any subsequent attempts to use countID will fail.

51074. The TPM invalidates sessions

5108 a. MUST invalidate all OSAP sessions associated with the counter

5109 b. MAY invalidate any other session

51105. If TPM_STCLEAR_DATA -> countID equals countID,

5111 a. Set TPM_STCLEAR_DATA -> countID to an illegal value (not the zero value)

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.

outputSize and outputData are always included in the outParamDigest. This includes stage 0, where the outputData contains the DAA session handle.

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 session handle used for owner authentication
		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 session digest for inputs and owner. HMAC key: ownerAuth.

5121 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
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DAA_Join.
4	4	3S	4	UINT32	outputSize	Size of outputData
5	<>	4S	<>	BYTE[]	outputData	Data produced by this capability
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 authHandle
7	1	4 H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20		20	TPM_AUTHDATA	resAuth	Authorization HMAC key: ownerAuth.

5122Description

5123This table summaries the input, output and saved data that is associated with each stage of
5124processing.

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	NULL	DAA_join_uo, DAA_join_u1	NULL	NULL
4	DAA_generic_R0	DAA_generic_n	P1=R0^f0 mod n	NULL	P1
5	DAA_generic_R1	DAA_generic_n	P2 = P1*(R1^f1) mod n	NULL	P2
6	DAA_generic_S0	DAA_generic_n	P3 = P2*(S0^u0) mod n	NULL	P3
7	DAA_generic_S1	DAA_generic_n	U = P3*(S1^u1) mod n	U	NULL
8	NE	NULL	U2	U2	NULL
9	DAA_generic_R0	DAA_generic_n	P1=R0^r0 mod n	NULL	P1
10	DAA_generic_R1	DAA_generic_n	P2 = P1*(R1^r1) mod n	NULL	P2
11	DAA_generic_S0	DAA_generic_n	P3 = P2*(S0^r2) mod n	NULL	P3
12	DAA_generic_S1	DAA_generic_n	P4 = P3*(S1^r3) mod n	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 mod q, E1 = w^r mod gamma	E1	NULL
16	c1	NULL	c = hash(c1 || NT)	nt	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	c	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

5125

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
3. Constant values of 0 or 1 are 1 byte integers, stages affected are
   a. 4(j), 5(j), 14(f), 17(e)
4. Representation of the strings “r0” to “r4” are 2-byte ASCII encodings, stages affected are
   a. 9(i), 10(h), 11(h), 12(h), 15(f),15(g), 17(d), 18(d), 19(d), 20(d), 21(d)

Start of informative comment:

5. Variable DAA_Count
   a. In stage 0, DAA_Count denotes the length of the RSA key chain, which certifies the main DAA public key and which will be loaded in stage 1. It also denotes the number of times stage 1 is executed.
   b. In stage 3 the variable DAA_count denotes the actual DAA counter. It allows a DAA issuer to keep track of the number of times it has issued 'different' DAA credentials to the same platform. (The counter does not need to be equal to the actual number.)

End of informative comment.

Stages

0. If stage==0
   c. Determine that sufficient resources are available to perform a TPM_DAA_Join.
      i. The TPM MUST support sufficient resources to perform one (1) TPM_DAA_Join/ TPM_DAA_Sign. The TPM MAY support additional TPM_DAA_Join/ TPM_DAA_Sign sessions.
      ii. The TPM may share internal resources between the DAA operations and other variable resource requirements:
         iii. If there are insufficient resources within the stored key pool (and one or more keys need to be removed to permit the DAA operation to execute) return TPM_NOSPACE
         iv. If there are insufficient resources within the stored session pool (and one or more authorization or transport sessions need to be removed to permit the DAA operation to execute), return TPM_RESOURCES.
   d. Set all fields in DAA_issuerSettings = NULL
   e. set all fields in DAA_tpmSpecific = NULL
   f. set all fields in DAA_session = NULL
   g. Set all fields in DAA_joinSession = NULL
   h. Verify that sizeOf(inputData0) == sizeOf(DAA_tpmSpecific -> DAA_count) and return error TPM_DAA_INPUT_DATA0 on mismatch

5163 i. Verify that inputData0 > 0, and return error TPM_DAA_INPUT_DATA0 on mismatch

5164 j. Set DAA_tpmSpecific -> DAA_count = inputData0

5165 k. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific ||
5166 DAA_joinSession)

5167 l. set DAA_session -> DAA_stage = 1

5168 m. Assign session handle for TPM_DAA_Join

5169 n. set outputData = new session handle

5170 i. The handle in outputData is included the output HMAC.

5171 o. return TPM_SUCCESS

51726. If stage==1

5173 a. Verify that DAA_session ->DAA_stage==1. Return TPM_DAA_STAGE and flush handle
5174 on mismatch

5175 b. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5176 DAA_joinSession) and return TPM_DAA_TPM_SETTINGS on mismatch

5177 c. Verify that sizeOf(inputData0) == DAA_SIZE_issuerModulus and return error
5178 TPM_DAA_INPUT_DATA0 on mismatch

5179 d. If DAA_session -> DAA_scratch == NULL:

5180 i. Set DAA_session -> DAA_scratch = inputData0

5181 ii. set DAA_joinSession -> DAA_digest_n0 = SHA-1(DAA_session -> DAA_scratch)

5182 iii. set DAA_tpmSpecific -> DAA_rekey = SHA-1(tpmDAASeed || DAA_joinSession
5183 -> DAA_digest_n0)

5184 e. Else (If DAA_session -> DAA_scratch != NULL):

5185 i. Set signedData = inputData0

5186 ii. Verify that sizeOf(inputData1) == DAA_SIZE_issuerModulus and return error
5187 TPM_DAA_INPUT_DATA1 on mismatch

5188 iii. Set signatureValue = inputData1

5189 iv. Use the RSA key == [DAA_session -> DAA_scratch] to verify that
5190 signatureValue is a signature on signedData using
5191 TPM_SS_RSASSAPKCS1v15_SHA1 (RSA PKCS1.5 with SHA-1), and return error
5192 TPM_DAA_ISSUER_VALIDITY on mismatch

5193 v. Set DAA_session -> DAA_scratch = signedData

5194 f. Decrement DAA_tpmSpecific -> DAA_count by 1 (unity)

5195 g. If DAA_tpmSpecific -> DAA_count == 0:

5196 i. increment DAA_session -> DAA_stage by 1

5197 h. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific ||
5198 DAA_joinSession)

5199 i. set outputData = NULL

j. return TPM_SUCCESS

17. 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 = (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 using TPM_SS_RSASSAPKCS1v15_SHA1 (RSA PKCS1.5 with SHA-1),, 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

18. 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. Obtain random data from the RNG and store it as DAA_joinSession -> DAA_join_u0

g. Obtain random data from the RNG and store it as DAA_joinSession -> DAA_join_u1

h. set outputData = NULL

i. increment DAA_session -> DAA_stage by 1

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

k. return TPM_SUCCESS

9. 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_DATA1 on mismatch

h. Set X = DAA_generic_R0

i. Set n = DAA_generic_n

j. Set f = SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0) || SHA-1(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

10. 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_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. Set X = DAA_generic_R1

i. Set n = DAA_generic_n

j. Set f = SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0) || SHA-1(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

11. 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_DATA1 on 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

12. If stage==7

5310 a. Verify that DAA_session ->DAA_stage==7. Return TPM_DAA_STAGE and flush handle
5311 on mismatch

5312 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5313 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5314 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5315 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

5316 d. Set DAA_generic_S1 = inputData0

5317 e. Verify that SHA-1(DAA_generic_S1) == DAA_issuerSettings -> DAA_digest_S1 and
5318 return error TPM_DAA_INPUT_DATA0 on mismatch

5319 f. Set DAA_generic_n = inputData1

5320 g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and
5321 return error TPM_DAA_INPUT_DATA1 on mismatch

5322 h. Set X = DAA_generic_S1

5323 i. Set n = DAA_generic_n

5324 j. Set Y = DAA_joinSession -> DAA_join_u1

5325 k. Set Z = DAA_session -> DAA_scratch

5326 l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n

5327 m. Set DAA_session -> DAA_digest to the SHA-1 (DAA_session -> DAA_scratch ||
5328 DAA_tpmSpecific -> DAA_count || DAA_joinSession -> DAA_digest_n0)

5329 n. set outputData = DAA_session -> DAA_scratch

5330 o. set DAA_session -> DAA_scratch = NULL

5331 p. increment DAA_session -> DAA_stage by 1

5332 q. return TPM_SUCCESS

533313. If stage==8

5334 a. Verify that DAA_session ->DAA_stage==8. Return TPM_DAA_STAGE and flush handle
5335 on mismatch

5336 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5337 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5338 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5339 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

5340 d. Verify inputSize0 == DAA_SIZE_NE and return error TPM_DAA_INPUT_DATA0 on
5341 mismatch

5342 e. Set NE = decrypt(inputData0, privEK)

5343 f. set outputData = SHA-1(DAA_session -> DAA_digest || NE)

5344 g. set DAA_session -> DAA_digest = NULL

5345 h. increment DAA_session -> DAA_stage by 1

5346 i. return TPM_SUCCESS

14. 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_DATA1 on mismatch
    h. Obtain random data from the RNG and store it as DAA_session -> DAA_contextSeed
    i. Obtain DAA_SIZE_r0 bytes using the MGF1 function and label them Y. “r0” || DAA_session -> DAA_contextSeed is the Z seed.
    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
15. 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_DATA1 on mismatch
    h. Obtain DAA_SIZE_r1 bytes using the MGF1 function and label them Y. “r1” || DAA_session -> DAA_contextSeed is the Z seed.
    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

16. 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 bytes using the MGF1 function and label them Y. “r2” || DAA_session -> DAA_contextSeed is the Z seed.

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

17. 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

5421 e. Verify that SHA-1(DAA_generic_S1) == DAA_issuerSettings -> DAA_digest_S1 and
5422 return error TPM_DAA_INPUT_DATA0 on mismatch

5423 f. Set DAA_generic_n = inputData1

5424 g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and
5425 return error TPM_DAA_INPUT_DATA1 on mismatch

5426 h. Obtain DAA_SIZE_r3 bytes using the MGF1 function and label them Y. “r3” ||
5427 DAA_session -> DAA_contextSeed is the Z seed.

5428 i. Set X = DAA_generic_S1

5429 j. Set n = DAA_generic_n

5430 k. Set Z = DAA_session -> DAA_scratch

5431 l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n

5432 m. set outputData = DAA_session -> DAA_scratch

5433 n. Set DAA_session -> DAA_scratch = NULL

5434 o. increment DAA_session -> DAA_stage by 1

5435 p. return TPM_SUCCESS

543618. If stage==13

5437 a. Verify that DAA_session->DAA_stage==13. Return TPM_DAA_STAGE and flush
5438 handle on mismatch

5439 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5440 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5441 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5442 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

5443 d. Set DAA_generic_gamma = inputData0

5444 e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings ->
5445 DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch

5446 f. Verify that inputSize1 == DAA_SIZE_w and return error TPM_DAA_INPUT_DATA1 on
5447 mismatch

5448 g. Set w = inputData1

5449 h. Set w1 = w^(DAA_issuerSettings -> DAA_generic_q) mod (DAA_generic_gamma)

5450 i. If w1 != 1 (unity), return error TPM_DAA_WRONG_W

5451 j. Set DAA_session -> DAA_scratch = w

5452 k. set outputData = NULL

5453 l. increment DAA_session -> DAA_stage by 1

5454 m. return TPM_SUCCESS.

545519. If stage==14

5456 a. Verify that DAA_session ->DAA_stage==14. Return TPM_DAA_STAGE and flush
5457 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 = SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 0) || SHA-1(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.

20. 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 bytes using the MGF1 function and label them r0. “r0” || DAA_session -> DAA_contextSeed is the Z seed.

g. Obtain DAA_SIZE_r1 bytes using the MGF1 function and label them r1. “r1” || DAA_session -> DAA_contextSeed is the Z seed.

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.

21. If stage==16

a. Verify that DAA_session ->DAA_stage==16. Return TPM_DAA_STAGE and flush handle on mismatch

5495 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5496 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5497 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5498 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

5499 d. Verify that inputSize0 == sizeOf(TPM_DIGEST) and return error
5500 TPM_DAA_INPUT_DATA0 on mismatch

5501 e. Set DAA_session -> DAA_digest = inputData0

5502 f. Obtain DAA_SIZE_NT bytes from the RNG and label them NT

5503 g. Set DAA_session -> DAA_digest to the SHA-1 (DAA_session -> DAA_digest || NT)

5504 h. Set outputData = NT

5505 i. increment DAA_session -> DAA_stage by 1

5506 j. return TPM_SUCCESS.

550722. If stage==17

5508 a. Verify that DAA_session ->DAA_stage==17. Return TPM_DAA_STAGE and flush
5509 handle on mismatch

5510 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5511 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5512 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5513 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

5514 d. Obtain DAA_SIZE_r0 bytes using the MGF1 function and label them r0. “r0” ||
5515 DAA_session -> DAA_contextSeed is the Z seed.

5516 e. Set f = SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count ||
5517 0) || SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1)
5518 mod DAA_issuerSettings -> DAA_generic_q.

5519 f. Set f0 = f mod 2^DAA_power0 (erase all but the lowest DAA_power0 bits of f)

5520 g. Set s0 = r0 + (DAA_session -> DAA_digest) * f0 in ***Z.*** Compute over the integers. The
5521 computation is not reduced with a modulus.

5522 h. set outputData = s0

5523 i. increment DAA_session -> DAA_stage by 1

5524 j. return TPM_SUCCESS

552523. If stage==18

5526 a. Verify that DAA_session ->DAA_stage==18. Return TPM_DAA_STAGE and flush
5527 handle on mismatch

5528 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5529 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5530 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5531 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

5532 d. Obtain DAA_SIZE_r1 bytes using the MGF1 function and label them r1. “r1” ||
5533 DAA_session -> DAA_contextSeed is the Z seed.

5534 e. Set f = SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count ||
5535 0) || SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1)
5536 mod DAA_issuerSettings -> DAA_generic_q.

5537 f. Shift f right by DAA_power0 bits (discard the lowest DAA_power0 bits) and label the
5538 result f1

5539 g. Set s1 = r1 + (DAA_session -> DAA_digest)* f1 in ***Z.*** Compute over the integers. The
5540 computation is not reduced with a modulus.

5541 h. set outputData = s1

5542 i. increment DAA_session -> DAA_stage by 1

5543 j. return TPM_SUCCESS

554424. If stage==19

5545 a. Verify that DAA_session ->DAA_stage==19. Return TPM_DAA_STAGE and flush
5546 handle on mismatch

5547 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5548 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5549 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5550 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

5551 d. Obtain DAA_SIZE_r2 bytes using the MGF1 function and label them r2. “r2” ||
5552 DAA_session -> DAA_contextSeed is the Z seed.

5553 e. Set s2 = r2 + (DAA_session -> DAA_digest)*(DAA_joinSession -> DAA_join_u0) mod
5554 2^DAA_power1 (Erase all but the lowest DAA_power1 bits of s2)

5555 f. set outputData = s2

5556 g. increment DAA_session -> DAA_stage by 1

5557 h. return TPM_SUCCESS

555825. If stage==20

5559 a. Verify that DAA_session ->DAA_stage==20. Return TPM_DAA_STAGE and flush
5560 handle on mismatch

5561 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5562 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5563 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5564 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

5565 d. Obtain DAA_SIZE_r2 bytes using the MGF1 function and label them r2. “r2” ||
5566 DAA_session -> DAA_contextSeed is the Z seed.

5567 e. Set s12 = r2 + (DAA_session -> DAA_digest)*(DAA_joinSession -> DAA_join_u0)

5568 f. Shift s12 right by DAA_power1 bit (discard the lowest DAA_power1 bits).

5569 g. Set DAA_session -> DAA_scratch = s12

5570 h. Set outputData = DAA_session -> DAA_digest

5571 i. increment DAA_session -> DAA_stage by 1

5572 j. return TPM_SUCCESS

557326. If stage==21

5574 a. Verify that DAA_session ->DAA_stage==21. Return TPM_DAA_STAGE and flush
5575 handle on mismatch

5576 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5577 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5578 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5579 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

5580 d. Obtain DAA_SIZE_r3 bytes using the MGF1 function and label them r3. “r3” ||
5581 DAA_session -> DAA_contextSeed is the Z seed.

5582 e. Set s3 = r3 + (DAA_session -> DAA_digest)*(DAA_joinSession -> DAA_join_u1) +
5583 (DAA_session -> DAA_scratch).

5584 f. Set DAA_session -> DAA_scratch = NULL

5585 g. set outputData = s3

5586 h. increment DAA_session -> DAA_stage by 1

5587 i. return TPM_SUCCESS

558827. If stage==22

5589 a. Verify that DAA_session ->DAA_stage==22. Return TPM_DAA_STAGE and flush
5590 handle on mismatch

5591 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5592 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5593 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5594 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

5595 d. Verify inputSize0 == DAA_SIZE_v0 and return error TPM_DAA_INPUT_DATA0 on
5596 mismatch

5597 e. Set u2 = inputData0

5598 f. Set v0 = u2 + (DAA_joinSession -> DAA_join_u0) mod 2^DAA_power1 (Erase all but
5599 the lowest DAA_power1 bits of v0).

5600 g. Set DAA_tpmSpecific -> DAA_digest_v0 = SHA-1(v0)

5601 h. Set v10 = u2 + (DAA_joinSession -> DAA_join_u0) in ***Z.*** Compute over the integers.
5602 The computation is not reduced with a modulus.

5603 i. Shift v10 right by DAA_power1 bits (erase the lowest DAA_power1 bits).

5604 j. Set DAA_session ->DAA_scratch = v10

5605 k. Set outputData

5606 i. Fill in TPM_DAA_BLOB with a type of TPM_RT_DAA_V0 and encrypt the v0
5607 parameters using TPM_PERMANENT_DATA -> daaBlobKey

5608 ii. set outputData to the encrypted TPM_DAA_BLOB

5609 l. increment DAA_session -> DAA_stage by 1

5610 m. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific ||
5611 DAA_joinSession)

5612 n. return TPM_SUCCESS

561328. If stage==23

5614 a. Verify that DAA_session ->DAA_stage==23. Return TPM_DAA_STAGE and flush
5615 handle on mismatch

5616 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5617 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5618 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5619 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

5620 d. Verify inputSize0 == DAA_SIZE_v1 and return error TPM_DAA_INPUT_DATA0 on
5621 mismatch

5622 e. Set u3 = inputData0

5623 f. Set v1 = u3 + DAA_joinSession -> DAA_join_u1 + DAA_session ->DAA_scratch

5624 g. Set DAA_tpmSpecific -> DAA_digest_v1 = SHA-1(v1)

5625 h. Set outputData

5626 i. Fill in TPM_DAA_BLOB with a type of TPM_RT_DAA_V1 and encrypt the v1
5627 parameters using TPM_PERMANENT_DATA -> daaBlobKey

5628 ii. set outputData to the encrypted TPM_DAA_BLOB

5629 i. Set DAA_session ->DAA_scratch = NULL

5630 j. increment DAA_session -> DAA_stage by 1

5631 k. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific ||
5632 DAA_joinSession)

5633 l. return TPM_SUCCESS

563429. If stage==24

5635 a. Verify that DAA_session ->DAA_stage==24. Return TPM_DAA_STAGE and flush
5636 handle on mismatch

5637 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5638 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5639 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5640 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

5641 d. set outputData = enc(DAA_tpmSpecific) using TPM_PERMANENT_DATA ->
5642 daaBlobKey

5643 e. Terminate the DAA session and all resources associated with the DAA join session
5644 handle.

5645 f. return TPM_SUCCESS

564630. If stage > 24, return error: TPM_DAA_STAGE

5647 26.2 TPM_DAA_Sign

5648**Start of informative comment:**

5649outputSize and outputData are always included in the outParamDigest. This includes stage 56500, where the outputData contains the DAA session handle.

5651 **End of informative comment.**

5652TPM protected capability; user must provide authorizations from the TPM Owner.

5653Incoming 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 session handle used for owner authentication
		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 session digest for inputs and owner. HMAC key: ownerAuth.

5654Outgoing 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
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal:TPM_ORD_DAA_Sign
4	4	3S	4	UINT32	outputSize	Size of outputData
5	<>	4S	<>	BYTE[]	outputData	Data produced by this capability
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 authHandle
7	1	4 H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
8	20		20	TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: ownerAuth.

5655Description

5656This table summaries the input, output and saved data that is associated with each stage of 5657processing.

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 mod n	NULL	P1
3	DAA_generic_R1	DAA_generic_n	P2 = P1*(R1^r1) mod n	NULL	P2
4	DAA_generic_S0	DAA_generic_n	P3 = P2*(S0^r2) mod n	NULL	P3
5	DAA_generic_S1	DAA_generic_n	T = P3*(S1^r4) mod n	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 mod q, 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) or 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 mod 2^power1	s2	NULL
14	enc(v0)	NULL	s12 = r2 + c*v0 >> power1	NULL	s12
15	enc(v1)	NULL	s3 = r4 + c*v1 + s12	s3	NULL

5658

5659When a TPM receives an Owner authorized command to input enc(DAA_tpmSpecific) or 5660enc(v0) or enc(v1), the TPM MUST verify that the TPM created the data and that neither the 5661data nor the TPM's daaProof has been changed since the data was created. Loading one of 5662these wrapped blobs does not require authorization, since correct blobs were created by the 5663TPM under Owner authorization, and unwrapped blobs cannot be used without Owner 5664authorisation. The TPM MUST NOT restrict the number of times that the contents of 5665enc(DAA_tpmSpecific) or enc(v0) or enc(v1) can be used by the same combination of TPM 5666and daaProof that created them.

5667Actions

5668A Trusted Platform Module that receives a valid TPM_DAA_Sign command SHALL:

56691. Use ownerAuth to verify that the Owner authorized all TPM_DAA_Sign input parameters.

56702. Any error results in the TPM invalidating all resources associated with the command

56713. Constant values of 0 or 1 are 1 byte integers, stages affected are

5672 a. 7(f), 11(e), 12(e)

56734. Representation of the strings “r0” to “r4” are 2-byte ASCII encodings, stages affected are

5674 a. 2(h), 3(h), 4(h), 5(h), 12(d), 13(f), 14(f), 15(f)

5675**Stages**

56760. If stage==0

5677 b. Determine that sufficient resources are available to perform a TPM_DAA_Sign.

5678 i. The TPM MUST support sufficient resources to perform one (1)
5679 TPM_DAA_Join/ TPM_DAA_Sign. The TPM MAY support addition TPM_DAA_Join/
5680 TPM_DAA_Sign sessions.

5681 ii. The TPM may share internal resources between the DAA operations and other
5682 variable resource requirements:

5683 iii. If there are insufficient resources within the stored key pool (and one or more
5684 keys need to be removed to permit the DAA operation to execute) return
5685 TPM_NOSPACE

5686 iv. If there are insufficient resources within the stored session pool (and one or
5687 more authorization or transport sessions need to be removed to permit the DAA
5688 operation to execute), return TPM_RESOURCES.

5689 c. Set DAA_issuerSettings = inputData0

5690 d. Verify that all fields in DAA_issuerSettings are present and return error
5691 TPM_DAA_INPUT_DATA0 if not.

5692 e. set all fields in DAA_session = NULL

5693 f. Assign new handle for session

5694 g. Set outputData to new handle

5695 i. The handle in outputData is included the output HMAC.

5696 h. set DAA_session -> DAA_stage = 1

5697 i. return TPM_SUCCESS

56985. If stage==1

5699 a. Verify that DAA_session ->DAA_stage==1. Return TPM_DAA_STAGE and flush handle
5700 on mismatch

5701 b. Set DAA_tpmSpecific = unwrap(inputData0) using TPM_PERMANENT_DATA ->
5702 daaBlobKey

5703 c. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5704 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5705 d. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific)

5706 e. set outputData = NULL

5707 f. set DAA_session -> DAA_stage =2

5708 g. return TPM_SUCCESS

57096. If stage==2

5710 a. Verify that DAA_session ->DAA_stage==2. Return TPM_DAA_STAGE and flush handle
5711 on mismatch

5712 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5713 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5714 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and
5715 return error TPM_DAA_TPM_SETTINGS on mismatch

5716 d. Set DAA_generic_R0 = inputData0

5717 e. Verify that SHA-1(DAA_generic_R0) == DAA_issuerSettings -> DAA_digest_R0 and
5718 return error TPM_DAA_INPUT_DATA0 on mismatch

5719 f. Set DAA_generic_n = inputData1

5720 g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and
5721 return error TPM_DAA_INPUT_DATA1 on mismatch

5722 h. Obtain random data from the RNG and store it as DAA_session -> DAA_contextSeed

5723 i. Obtain DAA_SIZE_r0 bytes using the MGF1 function and label them Y. “r0” ||
5724 DAA_session -> DAA_contextSeed is the Z seed.

5725 j. Set X = DAA_generic_R0

5726 k. Set n = DAA_generic_n

5727 l. Set DAA_session -> DAA_scratch = (X^Y) mod n

5728 m. set outputData = NULL

5729 n. increment DAA_session -> DAA_stage by 1

5730 o. return TPM_SUCCESS

57317. If stage==3

5732 a. Verify that DAA_session ->DAA_stage==3. Return TPM_DAA_STAGE and flush handle
5733 on mismatch

5734 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5735 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5736 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and
5737 return error TPM_DAA_TPM_SETTINGS on mismatch

5738 d. Set DAA_generic_R1 = inputData0

5739 e. Verify that SHA-1(DAA_generic_R1) == DAA_issuerSettings -> DAA_digest_R1 and
5740 return error TPM_DAA_INPUT_DATA0 on mismatch

5741 f. Set DAA_generic_n = inputData1

5742 g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and
5743 return error TPM_DAA_INPUT_DATA1 on mismatch

5744 h. Obtain DAA_SIZE_r1 bytes using the MGF1 function and label them Y. “r1” ||
5745 DAA_session -> DAA_contextSeed is the Z seed.

5746 i. Set X = DAA_generic_R1

5747 j. Set n = DAA_generic_n

5748 k. Set Z = DAA_session -> DAA_scratch

5749 l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n

5750 m. set outputData = NULL

5751 n. increment DAA_session -> DAA_stage by 1

5752 o. return TPM_SUCCESS

57538. If stage==4

5754 a. Verify that DAA_session ->DAA_stage==4. Return TPM_DAA_STAGE and flush handle
5755 on mismatch

5756 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5757 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5758 c. Verify that DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific) and
5759 return error TPM_DAA_TPM_SETTINGS on mismatch

5760 d. Set DAA_generic_S0 = inputData0

5761 e. Verify that SHA-1(DAA_generic_S0) == DAA_issuerSettings -> DAA_digest_S0 and
5762 return error TPM_DAA_INPUT_DATA0 on mismatch

5763 f. Set DAA_generic_n = inputData1

5764 g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and
5765 return error TPM_DAA_INPUT_DATA1 on mismatch

5766 h. Obtain DAA_SIZE_r2 bytes using the MGF1 function and label them Y. “r2” ||
5767 DAA_session -> DAA_contextSeed is the Z seed.

5768 i. Set X = DAA_generic_S0

5769 j. Set n = DAA_generic_n

5770 k. Set Z = DAA_session -> DAA_scratch

5771 l. Set DAA_session -> DAA_scratch = Z*(X^Y) mod n

5772 m. set outputData = NULL

5773 n. increment DAA_session -> DAA_stage by 1

5774 o. return TPM_SUCCESS

57759. If stage==5

5776 a. Verify that DAA_session ->DAA_stage==5. Return TPM_DAA_STAGE and flush handle
5777 on mismatch

5778 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5779 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5780 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and
5781 return error TPM_DAA_TPM_SETTINGS on mismatch

5782 d. Set DAA_generic_S1 = inputData0

5783 e. Verify that SHA-1(DAA_generic_S1) == DAA_issuerSettings -> DAA_digest_S1 and
5784 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 bytes using the MGF1 function and label them Y. “r4” || DAA_session -> DAA_contextSeed is the Z seed.

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

10. 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.

11. 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

5822 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and
5823 return error TPM_DAA_TPM_SETTINGS on mismatch

5824 d. Set DAA_generic_gamma = inputData0

5825 e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings ->
5826 DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch

5827 f. Set f = SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count ||
5828 0) || SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1)
5829 mod DAA_issuerSettings -> DAA_generic_q.

5830 g. Set E = ((DAA_session -> DAA_scratch)^f) mod (DAA_generic_gamma).

5831 h. Set outputData = E

5832 i. increment DAA_session -> DAA_stage by 1

5833 j. return TPM_SUCCESS.

583412. If stage==8

5835 a. Verify that DAA_session ->DAA_stage==8. Return TPM_DAA_STAGE and flush handle
5836 on mismatch

5837 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5838 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5839 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and
5840 return error TPM_DAA_TPM_SETTINGS on mismatch

5841 d. Set DAA_generic_gamma = inputData0

5842 e. Verify that SHA-1(DAA_generic_gamma) == DAA_issuerSettings ->
5843 DAA_digest_gamma and return error TPM_DAA_INPUT_DATA0 on mismatch

5844 f. Obtain DAA_SIZE_r0 bytes using the MGF1 function and label them r0. “r0” ||
5845 DAA_session -> DAA_contextSeed is the Z seed.

5846 g. Obtain DAA_SIZE_r1 bytes using the MGF1 function and label them r1. “r1” ||
5847 DAA_session -> DAA_contextSeed is the Z seed.

5848 h. set r = r0 + 2^DAA_power0 * r1 mod (DAA_issuerSettings -> DAA_generic_q).

5849 i. Set E1 = ((DAA_session -> DAA_scratch)^r) mod (DAA_generic_gamma)

5850 j. Set DAA_session -> DAA_scratch = NULL

5851 k. Set outputData = E1

5852 l. increment DAA_session -> DAA_stage by 1

5853 m. return TPM_SUCCESS.

585413. If stage==9

5855 a. Verify that DAA_session ->DAA_stage==9. Return TPM_DAA_STAGE and flush handle
5856 on mismatch

5857 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5858 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 bytes 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.

14. 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. Verify that inputSize0 == sizeOf(BYTE), and return error TPM_DAA_INPUT_DATA0 on mismatch

e. Set selector = inputData0, verify that selector == 0 or 1, and return error TPM_DAA_INPUT_DATA0 on mismatch

f. If selector == 1, verify that inputSize1 == sizeOf(TPM_DIGEST), and return error TPM_DAA_INPUT_DATA1 on mismatch

g. Set DAA_session -> DAA_digest to SHA-1 (DAA_session -> DAA_digest || 1 || inputData1)

h. If selector == 0, verify that inputData1 is a handle to a TPM identity key (AIK), and return error TPM_DAA_INPUT_DATA1 on mismatch

i. Set DAA_session -> DAA_digest to SHA-1 (DAA_session -> DAA_digest || 0 || n2) where n2 is the modulus of the AIK

j. Set outputData = DAA_session -> DAA_digest

k. increment DAA_session -> DAA_stage by 1

l. return TPM_SUCCESS.

15. 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

5896 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and
5897 return error TPM_DAA_TPM_SETTINGS on mismatch

5898 d. Obtain DAA_SIZE_r0 bytes using the MGF1 function and label them r0. “r0” ||
5899 DAA_session -> DAA_contextSeed is the Z seed.

5900 e. Set f = SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count ||
5901 0) || SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1)
5902 mod DAA_issuerSettings -> DAA_generic_q.

5903 f. Set f0 = f mod 2^DAA_power0 (erase all but the lowest DAA_power0 bits of f)

5904 g. Set s0 = r0 + (DAA_session -> DAA_digest)*(f0)

5905 h. set outputData = s0

5906 i. increment DAA_session -> DAA_stage by 1

5907 j. return TPM_SUCCESS

590816. If stage==12

5909 a. Verify that DAA_session ->DAA_stage==12. Return TPM_DAA_STAGE and flush
5910 handle on mismatch

5911 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5912 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5913 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and
5914 return error TPM_DAA_TPM_SETTINGS on mismatch

5915 d. Obtain DAA_SIZE_r1 bytes using the MGF1 function and label them r1. “r1” ||
5916 DAA_session -> DAA_contextSeed is the Z seed.

5917 e. Set f = SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count ||
5918 0) || SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1)
5919 mod DAA_issuerSettings -> DAA_generic_q.

5920 f. Shift f right by DAA_power0 bits (discard the lowest DAA_power0 bits) and label the
5921 result f1

5922 g. Set s1 = r1 + (DAA_session -> DAA_digest)*(f1)

5923 h. set outputData = s1

5924 i. increment DAA_session -> DAA_stage by 1

5925 j. return TPM_SUCCESS

592617. If stage==13

5927 a. Verify that DAA_session ->DAA_stage==13. Return TPM_DAA_STAGE and flush
5928 handle on mismatch

5929 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5930 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5931 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and
5932 return error TPM_DAA_TPM_SETTINGS on mismatch

5933 d. Set DAA_private_v0= unwrap(inputData0) using TPM_PERMANENT_DATA ->
5934 daaBlobKey

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 bytes from the MGF1 function and label them r2. “r2” || DAA_session -> DAA_contextSeed is the Z seed.

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 outputData = s2

i. increment DAA_session -> DAA_stage by 1

j. return TPM_SUCCESS

18. 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) and return error TPM_DAA_TPM_SETTINGS on mismatch

d. Set DAA_private_v0= unwrap(inputData0) using TPM_PERMANENT_DATA -> daaBlobKey

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 bytes using the MGF1 function and label them r2. “r2” || DAA_session -> DAA_contextSeed is the Z seed.

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

19. 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) using TPM_PERMANENT_DATA -> daaBlobKey

5972 e. Verify that SHA-1(DAA_private_v1) == DAA_tpmSpecific -> DAA_digest_v1 and return
5973 error TPM_DAA_INPUT_DATA0 on mismatch

5974 f. Obtain DAA_SIZE_r4 bytes using the MGF1 function and label them r4. “r4” ||
5975 DAA_session -> DAA_contextSeed is the Z seed.

5976 g. Set s3 = r4 + (DAA_session -> DAA_digest)*(DAA_private_v1) + (DAA_session ->
5977 DAA_scratch).

5978 h. Set DAA_session -> DAA_scratch = NULL

5979 i. set outputData = s3

5980 j. Terminate the DAA session and all resources associated with the DAA sign session
5981 handle.

5982 k. return TPM_SUCCESS

598320. If stage > 15, return error: TPM_DAA_STAGE

27. 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

27.1 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_Terminate_Handle by TPM_FlushSpecific.

End of informative comment.

27.1.1 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_EvictKey

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.

New 1.2 functionality

The command must check the status of the ownerEvict flag for the key and if the flag is TRUE return TPM_KEY_CONTROL_OWNER

6010 27.1.2 TPM_Terminate_Handle

6011 **Start of informative comment:**

6012 This allows the TPM manager to clear out information in a session handle.

6013 The TPM may maintain the authorization session even though a key attached to it has been
6014 unloaded or the authorization session itself has been unloaded in some way. When a
6015 command is executed that requires this session, it is the responsibility of the external
6016 software to load both the entity and the authorization session information prior to
6017 command execution.

6018 **End of informative comment.**

6019 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Terminate_Handle.
4	4			TPM_AUTHHANDLE	handle	The handle to terminate

6020 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Terminate_Handle.

6021 Description

6022 The TPM SHALL terminate the session and destroy all data associated with the session
6023 indicated.

6024 Actions

6025 A TPM SHALL unilaterally perform the actions of TPM_Terminate_Handle upon detection of
6026 the following events:

6027 1. Completion of a received command whose authorization “continueUse” flag is FALSE.

6028 2. Completion of a received command when a shared secret derived from the authorization
6029 session was exclusive-or’ed with data (to provide confidentiality for that data). This
6030 occurs during execution of a TPM_ChangeAuth command, for example.

6031 3. When the associated entity is destroyed (in the case of TPM Owner or SRK, for example)

6032 4. Upon execution of TPM_Init

60335. When the command returns an error. This is due to the fact that when returning an 6034 error the TPM does not send back nonceEven. There is no way to maintain the rolling 6035 nonces, hence the TPM MUST terminate the authorization session.

60366. Failure of an authorization check belonging to that authorization session.

6037 27.2 Context management

6038**Start of informative comment:**

6039The 1.1 context commands were written for specific resource types. The 1.2 commands are
6040generic for all resource types. So the Savexxx commands are replaced by TPM_SaveContext
6041and the LoadXXX commands by TPM_LoadContext.

6042**End of informative comment.**

6043 27.2.1 TPM_SaveKeyContext

6044**Start of informative comment:**

6045TPM_SaveKeyContext saves a loaded key outside the TPM. After creation of the key context
6046blob the TPM automatically releases the internal memory used by that key. The format of
6047the key context blob is specific to a TPM.

6048**End of informative comment.**

6049 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveKeyContext
4	4			TPM_KEY_HANDLE	keyHandle	The key which will be kept outside the TPM

6050 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveKeyContext
4	4	3S	4	UINT32	keyContextSize	The actual size of the outgoing key context blob. If the command fails the value will be 0
5	<>	4S	<>	BYTE[]	keyContextBlob	The key context blob.

6051 Description

60521. This command allows saving a loaded key outside the TPM. After creation of the
6053 keyContextBlob, the TPM automatically releases the internal memory used by that key.
6054 The format of the key context blob is specific to a TPM.

60552. A TPM protected capability belonging to the TPM that created a key context blob MUST
6056 be the only entity that can interpret the contents of that blob. If a cryptographic
6057 technique is used for this purpose, the level of security provided by that technique
6058 SHALL be at least as secure as a 2048 bit RSA algorithm. Any secrets (such as keys)

6059 used in such a cryptographic technique MUST be generated using the TPM's random
6060 number generator. Any symmetric key MUST be used within the power-on session
6061 during which it was created, only.

60623. A key context blob SHALL enable verification of the integrity of the contents of the blob
6063 by a TPM protected capability.

60644. A key context blob SHALL enable verification of the session validity of the contents of the
6065 blob by a TPM protected capability. The method SHALL ensure that all key context blobs
6066 are rendered invalid if power to the TPM is interrupted.

27.2.2 TPM_LoadKeyContext

Start of informative comment:

TPM_LoadKeyContext loads 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.

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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadKeyContext
4	4	2S	4	UINT32	keyContextSize	The size of the following key context blob.
5	<>	3S	<>	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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadKeyContext
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.

27.2.3 TPM_SaveAuthContext

Start of informative comment:

TPM_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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveAuthContext
4	4			TPM_AUTHHANDLE	authHandle	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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SaveAuthContext
4	4	3S	4	UINT32	authContextSize	The actual size of the outgoing authorization context blob. If the command fails the value will be 0.
5	<>	4S	4	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.

27.2.4 TPM_LoadAuthContext

Start of informative comment:

TPM_LoadAuthContext loads 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.

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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadAuthContext
4	4	2S	4	UINT32	authContextSize	The size of the following authorization context blob.
5	<>	3S	<>	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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_LoadAuthContext
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.

For an OSAP authorization context blob referring to a key, verify that the key linked to this session is resident in the TPM.

6131

27.3 DIR commands

Start of informative comment:

The DIR commands are replaced by the NV storage commands.

The DIR [0] in 1.1 is now TPM_PERMANENT_DATA -> authDIR[0] and is always available for the TPM to use. It is accessed by DIR commands using dirIndex 0 and by NV commands using nvIndex TPM_NV_INDEX_DIR.

If the TPM vendor supports additional DIR registers, the TPM vendor may return errors or provide vendor specific mappings for those DIR registers to NV storage locations.

End of informative comment.

1. A dirIndex value of 0 MUST corresponds to an NV storage nvIndex value TPM_NV_INDEX_DIR.
2. The TPM vendor MAY return errors or MAY provide vendor specific mappings for DIR dirIndex values greater than 0 to NV storage locations.

27.3.1 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.

Access is also provided through the NV commands with nvIndex TPM_NV_INDEX_DIR. Owner authorization is not required when nvLocked is FALSE.

Version 1.2 requires only one DIR. If the DIR named does not exist, the TPM_DirWriteAuth 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 session 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 session handle
9	20			TPM_AUTHDATA	ownerAuth	The authorization session 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.
		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 session digest for the returned parameters. HMAC key: ownerAuth.

Actions

61581. Validate that authHandle contains a TPM Owner AuthData to execute the 6159 TPM_DirWriteAuth command

61602. Validate that dirIndex points to a valid DIR on this TPM

61613. Write newContents into the DIR pointed to by dirIndex

27.3.2 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 AuthData is available early in boot. TSS implementers may choose to provide other means of authorizing this action. Version 1.2 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DirRead.
4	4	2S	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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DirRead.
4	20	3S	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

6176 27.4 Change Auth

6177**Start of informative comment:**

6178The change auth commands can be duplicated by creating a transport session with
6179confidentiality and issuing the changeAuth command.

6180**End of informative comment.**

27.4.1 TPM_ChangeAuthAsymStart

Start of informative comment:

The TPM_ChangeAuthAsymStart starts the process of changing AuthData 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 AuthData 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 AuthData 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 AuthData.

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 session 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 session handle
10	20			TPM_AUTHDATA	idAuth	Authorization. HMAC key: idKey.usageAuth.

6204Outgoing 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.

6205Actions

62061. The TPM SHALL verify the AuthData to use the TPM identity key held in idHandle. The 6207 TPM MUST verify that the key is a TPM identity key.

62082. The TPM SHALL validate the algorithm parameters for the key to create from the 6209 tempKey parameter.

62103. Recommended key type is RSA

62114. Minimum RSA key size MUST is 512 bits, recommended RSA key size is 1024

62125. For other key types the minimum key size strength MUST be comparable to RSA 512

62136. If the TPM is not designed to create a key of the requested type, return the error code 6214 TPM_BAD_KEY_PROPERTY

62157. The TPM SHALL create a new key (k1) in accordance with the algorithm parameter. The 6216 newly created key is pointed to by ephHandle.

62178. The TPM SHALL fill in all fields in tempKey using k1 for the information. The TPM_KEY 6218 -> encSize MUST be 0.

62199. The TPM SHALL fill in certifyInfo using k1 for the information. The certifyInfo -> data 6220 field is supplied by the antiReplay.

622110. The TPM then signs the certifyInfo parameter using the key pointed to by idHandle. The 6222 resulting signed blob is returned in sig parameter

6223Field Descriptions for certifyInfo parameter

Type	Name	Description
TPM_VERSION	Version	TPM version structure; Part 2 TPM_VERSION
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.

6224

6225 27.4.2 TPM_ChangeAuthAsymFinish

6226 **Start of informative comment:**

6227 The TPM_ChangeAuthAsymFinish command allows the owner of an entity to change the
6228 AuthData for the entity.

6229 The command requires the cooperation of the owner of the parent of the entity, since
6230 AuthData must be provided to use that parent entity. The command requires knowledge of
6231 the existing AuthData information and passes the new AuthData information. The
6232 newAuthLink parameter proves knowledge of existing AuthData information and new
6233 AuthData information. The new AuthData information "encNewAuth" is encrypted using the
6234 "tempKey" variable obtained via TPM_ChangeAuthAsymStart.

6235 A parent therefore retains control over a change in the AuthData of a child, but is prevented
6236 from knowing the new AuthData for that child.

6237 The changeProof parameter provides a proof that the new AuthData value was properly
6238 inserted into the entity. The inclusion of a nonce from the TPM provides an entropy source
6239 in the case where the AuthData value may be in itself be a low entropy value (hash of a
6240 password etc).

6241 **End of informative comment.**

6242 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 AuthData values together
8	4	5S	4	UINT32	newAuthSize	Size of encNewAuth
9	<>	6S	<>	BYTE[ ]	encNewAuth	New AuthData 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 session handle
15	20			TPM_AUTHDATA	privAuth	The authorization session digest for inputs and parentHandle. HMAC key: parentKey.usageAuth.

6243

6244Outgoing 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 AuthData 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 session digest for the returned parameters. HMAC key: parentKey.usageAuth.

6245Description

6246If the parentHandle points to the SRK then the HMAC key MUST be built using the TPM
6247Owner authentication.

6248Actions

62491. The TPM SHALL validate that the authHandle parameter authorizes use of the key in
6250 parentHandle.

62512. The encData field MUST be the encData field from TPM_STORED_DATA or TPM_KEY.

62523. The TPM SHALL create e1 by decrypting the entity held in the encData parameter.

62534. The TPM SHALL create a1 by decrypting encNewAuth using the ephHandle ->
6254 TPM_KEY_AUTHCHANGE private key. a1 is a structure of type
6255 TPM_CHANGEAUTH_VALIDATE.

62565. The TPM SHALL create b1 by performing the following HMAC calculation: b1 = HMAC
6257 (a1 -> newAuthSecret). The secret for this calculation is encData -> currentAuth. This
6258 means that b1 is a value built from the current AuthData value (encData ->
6259 currentAuth) and the new AuthData value (a1 -> newAuthSecret).

62606. The TPM SHALL compare b1 with newAuthLink. The TPM SHALL indicate a failure if the
6261 values do not match.

62627. The TPM SHALL replace e1 -> authData with a1 -> newAuthSecret

62638. The TPM SHALL encrypt e1 using the appropriate functions for the entity type. The key
6264 to encrypt with is parentHandle.

62659. 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.

27.5 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	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: 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	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Reset.

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_SaveAuthContext and TPM_SaveKeyContext
   b. The TPM MUST invalidate OSAP sessions
   c. The TPM MAY invalidate DSAP sessions
   d. The TPM MUST NOT invalidate structures created by TPM_SaveContext
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

6291 27.6 TPM_OwnerReadPubek

6292**Start of informative comment:**

6293Return the endorsement key public portion. This is authorized by the TPM Owner.

6294**End of informative comment.**

6295 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 session handle used for owner authentication.
		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 session handle
7	20			TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner authentication. HMAC key: ownerAuth.

6296 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 session digest for the returned parameters. HMAC key: ownerAuth.

6297 Description

6298This command returns the PUBEK.

6299 Actions

6300The TPM_OwnerReadPubek command SHALL

63011. Validate the TPM Owner AuthData to execute this command

63022. Export the PUBEK

27.7 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.

This command has in essence been deprecated as TPM_TakeOwnership now sets the value to false. The command remains at this time for backward compatibility.

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 session handle used for owner authentication.
		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 session handle
7	20			TPM_AUTHDATA	ownerAuth	The authorization session 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 session digest for the returned parameters. HMAC key: ownerAuth.

Actions

1. This capability sets the TPM_PERMANENT_FLAGS -> readPubek flag to FALSE.

6315 27.8 TPM_LoadKey

6316**Start of informative comment:**

6317Version 1.2 deprecates TPM_LoadKey due to the HMAC of the new key handle on return. 6318The wrapping makes use of the handle difficult in an environment where the TSS, or other 6319management entity, is changing the TPM handle to a virtual handle.

6320Software using TPM_LoadKey on a 1.2 TPM can have a collision with the returned handle as 6321the 1.2 TPM uses random values in the lower three bytes of the handle. All new software 6322must use LoadKey2 to allow management software the ability to manage the key handle.

6323Before the TPM can use a key to either wrap, unwrap, bind, unbind, seal, unseal, sign or 6324perform any other action, it needs to be present in the TPM. The TPM_LoadKey function 6325loads the key into the TPM for further use.

6326The TPM assigns the key handle. The TPM always locates a loaded key by use of the handle. 6327The assumption is that the handle may change due to key management operations. It is the 6328responsibility of upper level software to maintain the mapping between handle and any 6329label used by external software.

6330This command has the responsibility of enforcing restrictions on the use of keys. For 6331example, when attempting to load a STORAGE key it will be checked for the restrictions on 6332a storage key (2048 size etc.).

6333The load command must maintain a record of whether any previous key in the key 6334hierarchy was bound to a PCR using parentPCRStatus.

6335The flag parentPCRStatus enables the possibility of checking that a platform passed 6336through some particular state or states before finishing in the current state. A grandparent 6337key could be linked to state-1, a parent key could linked to state-2, and a child key could be 6338linked to state-3, for example. The use of the child key then indicates that the platform 6339passed through states 1 and 2 and is currently in state 3, in this example. TPM_Startup 6340with stType == TPM_ST_CLEAR indicates that the platform has been reset, so the platform 6341has not passed through the previous states. Hence keys with parentPCRStatus==TRUE 6342must be unloaded if TPM_Startup is issued with stType == TPM_ST_CLEAR.

6343If a TPM_KEY structure has been decrypted AND the integrity test using "pubDataDigest" 6344has passed AND the key is non-migratory, the key must have been created by the TPM. So 6345there is every reason to believe that the key poses no security threat to the TPM. While there 6346is no known attack from a rogue migratory key, there is a desire to verify that a loaded 6347migratory key is a real key, arising from a general sense of unease about execution of 6348arbitrary data as a key. Ideally a consistency check would consist of an encrypt/decrypt 6349cycle, but this may be expensive. For RSA keys, it is therefore suggested that the 6350consistency test consists of dividing the supposed RSA product by the supposed RSA prime, 6351and checking that there is no remainder.

6352**End of informative comment.**

6353Incoming 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_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 session 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 session handle
9	20			TPM_AUTHDATA	parentAuth	The authorization session digest for inputs and parentHandle. HMAC key: parentKey.usageAuth.

6354Outgoing 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 session digest for the returned parameters. HMAC key: parentKey.usageAuth.

6355Actions

6356The TPM SHALL perform the following steps:

63571. Validate the command and the parameters using parentAuth and parentHandle -> 6358 usageAuth

63592. If parentHandle -> keyUsage is NOT TPM_KEY_STORAGE return 6360 TPM_INVALID_KEYUSAGE

63613. If the TPM is not designed to operate on a key of the type specified by inKey, return the 6362 error code TPM_BAD_KEY_PROPERTY

63634. The TPM MUST handle both TPM_KEY and TPM_KEY12 structures

63645. Decrypt the inKey -> privkey to obtain TPM_STORE_ASYMKEY structure using the key 6365 in parentHandle

63666. Validate the integrity of inKey and decrypted TPM_STORE_ASYMKEY

6367 a. Reproduce inKey -> TPM_STORE_ASYMKEY -> pubDataDigest using the fields of
6368 inKey, and check that the reproduced value is the same as pubDataDigest

63697. Validate the consistency of the key and it's key usage.

6370 a. If inKey -> keyFlags -> migratable is TRUE, the TPM SHALL verify consistency of the
6371 public and private components of the asymmetric key pair. If inKey -> keyFlags ->
6372 migratable is FALSE, the TPM MAY verify consistency of the public and private
6373 components of the asymmetric key pair. The consistency of an RSA key pair MAY be
6374 verified by dividing the supposed (P*Q) product by a supposed prime and checking that
6375 there is no remainder.

6376 b. If inKey -> keyUsage is TPM_KEY_IDENTITY, verify that inKey->keyFlags->migratable
6377 is FALSE. If it is not, return TPM_INVALID_KEYUSAGE

6378 c. If inKey -> keyUsage is TPM_KEY_AUTHCHANGE, return TPM_INVALID_KEYUSAGE

6379 d. If inKey -> keyFlags -> migratable equals 0 then verify that TPM_STORE_ASYMKEY
6380 -> migrationAuth equals TPM_PERMANENT_DATA -> tpmProof

6381 e. Validate the mix of encryption and signature schemes

6382 f. If TPM_PERMANENT_FLAGS -> FIPS is TRUE then

6383 i. If keyInfo -> keySize is less than 1024 return TPM_NOTFIPS

6384 ii. If keyInfo -> authDataUsage specifies TPM_AUTH_NEVER return
6385 TPM_NOTFIPS

6386 iii. If keyInfo -> keyUsage specifies TPM_KEY_LEGACY return TPM_NOTFIPS

6387 g. If inKey -> keyUsage is TPM_KEY_STORAGE or TPM_KEY_MIGRATE

6388 i. algorithmID MUST be TPM_ALG_RSA

6389 ii. Key size MUST be 2048

6390 iii. exponentSize MUST be 0

6391 iv. sigScheme MUST be TPM_SS_NONE

6392 h. If inKey -> keyUsage is TPM_KEY_IDENTITY

6393 i. algorithmID MUST be TPM_ALG_RSA

6394 ii. Key size MUST be 2048

6395 iii. exponentSize MUST be 0

6396 iv. encScheme MUST be TPM_ES_NONE

6397 i. If the decrypted inKey -> pcrInfo is NULL,

6398 i. The TPM MUST set the internal indicator to indicate that the key is not using
6399 any PCR registers.

6400 j. Else

6401 i. The TPM MUST store pcrInfo in a manner that allows the TPM to calculate a
6402 composite hash whenever the key will be in use

6403 ii. The TPM MUST handle both version 1.1 TPM_PCR_INFO and 1.2
6404 TPM_PCR_INFO_LONG structures according to the type of TPM_KEY structure

6405 (1) The TPM MUST validate the TPM_PCR_INFO or TPM_PCR_INFO_LONG
6406 structures for legal values. However, the digestAtRelease and
6407 localityAtRelease are not validated for authorization until use time.

64088. Perform any processing necessary to make TPM_STORE_ASYMKEY key available for
6409 operations

64109. Load key and key information into internal memory of the TPM. If insufficient memory
6411 exists, return error TPM_NOSPACE.

641210. Assign inKeyHandle according to internal TPM rules.

641311. Set InKeyHandle -> parentPCRStatus to parentHandle -> parentPCRStatus.

641412. If parentHandle indicates it is using PCR registers, then set inKeyHandle ->
6415 parentPCRStatus to TRUE.

28. 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

6422 28.1 TPM_GetCapabilitySigned

6423**Start of informative comment:**

6424Along with TPM_GetCapabilityOwner this command allowed the possible signature of 6425improper values.

6426TPM_GetCapabilitySigned is almost the same as TPM_GetCapability. The differences are 6427that the input includes a challenge (a nonce) and the response includes a digital signature 6428to vouch for the source of the answer.

6429If a caller itself requires proof, it is sufficient to use any signing key for which only the TPM 6430and the caller have AuthData.

6431If a caller requires proof for a third party, the signing key must be one whose signature is 6432trusted by the third party. A TPM-identity key may be suitable.

6433**End of informative comment.**

6434 Deleted Ordinal

6435TPM_GetCapabilitySigned

6436 28.2 TPM_GetOrdinalAuditStatus

6437 **Start of informative comment:**

6438 Get the status of the audit flag for the given ordinal.

6439 **End of informative comment.**

6440 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

6441 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

6442 Actions

64431. The TPM returns the Boolean value for the given ordinal. The value is TRUE if the
6444 command is being audited.

6445 28.3 TPM_CertifySelfTest

6446**Start of informative comment:**

6447TPM_CertifySelfTest causes the TPM to perform a full self-test and return an authenticated 6448value if the test passes.

6449If a caller itself requires proof, it is sufficient to use any signing key for which only the TPM 6450and the caller have AuthData.

6451If a caller requires proof for a third party, the signing key must be one whose signature is 6452trusted by the third party. A TPM-identity key may be suitable.

6453**End of informative comment.**

6454 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	Anti Replay nonce to prevent replay of messages
6	4			TPM_AUTHHANDLE	authHandle	The authorization session 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 session handle
9	20			TPM_AUTHDATA	privAuth	The authorization session digest that authorizes the inputs and use of keyHandle. HMAC key: key.usageAuth

6455 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 session 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
   a. If the key pointed to by keyHandle has a signature scheme that is not TPM_SS_RSASSAPKCS1v15_SHA1, the TPM may either return TPM_BAD_SCHEME or may return TPM_SUCCESS and a vendor specific signature.
3. Create t1 the NOT null terminated string of "Test Passed", i.e. 11 bytes.
4. The TPM creates m2 the message to sign by concatenating t1 || AntiReplay || ordinal.
5. The TPM signs the SHA-1 of m2 using the key identified by keyHandle, and returns the signature as sig.

28.4 TPM_GetAuditEvent

Start of informative comment:

Deleted

End of informative comment.

Deleted Ordinal

TPM_GetAuditEvent

6481 28.5 TPM_GetAuditEventSigned

6482

6483**Start of informative comment:**

6484Deleted

6485**End of informative comment.**

6486 Deleted Ordinal

6487TPM_GetAuditEventSigned

6488

6489