

TPM Main Part 3 Commands

**Specification Version 1.2
Level 2 Revision 103
9 July 2007
Published**

Contact: tpmwg@trustedcomputinggroup.org

TCG Published

Copyright © TCG 2003-2007

TCG

Copyright © 2003-2005 Trusted Computing Group, Incorporated.

Disclaimer

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Without limitation, TCG disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

No license, express or implied, by estoppel or otherwise, to any TCG or TCG member intellectual property rights is granted herein.

Except that a license is hereby granted by TCG to copy and reproduce this specification for internal use only.

Contact the Trusted Computing Group at www.trustedcomputinggroup.org for information on specification licensing through membership agreements.

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

Acknowledgement

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

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

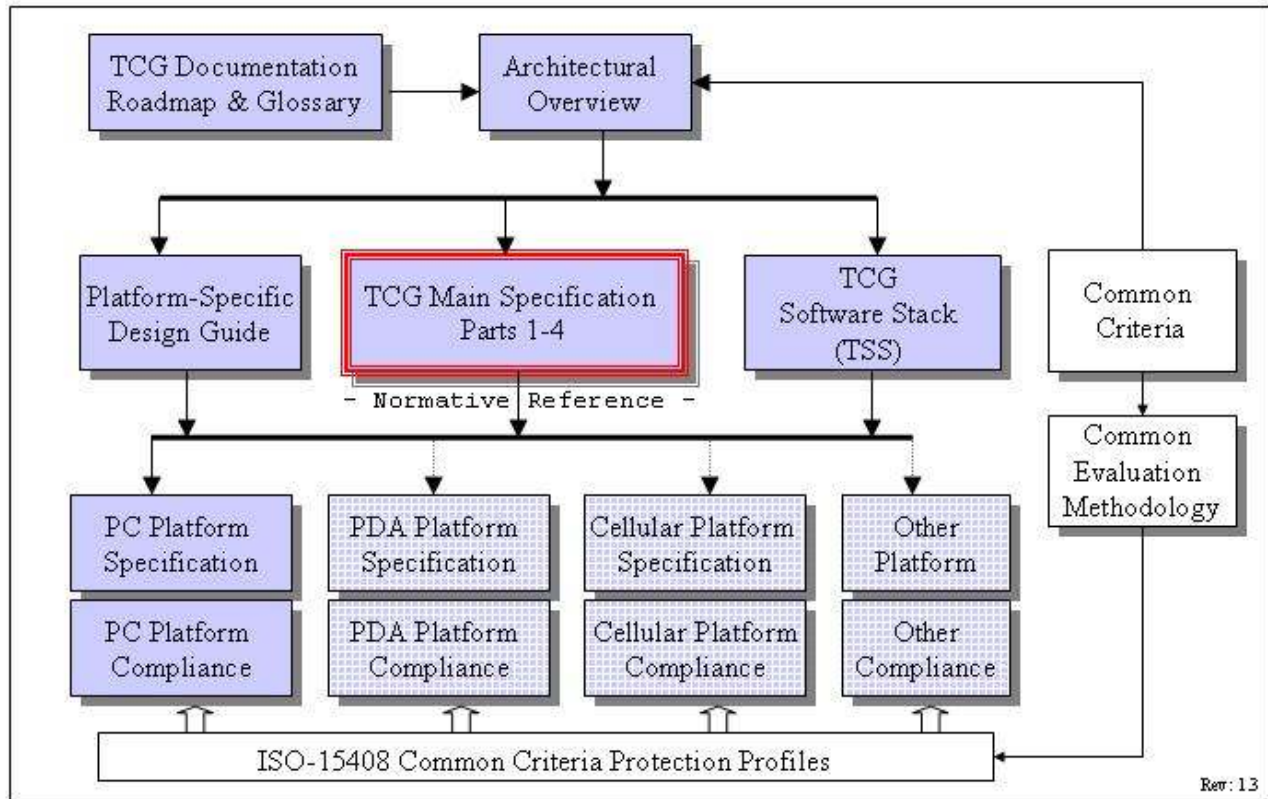
David Grawrock

TPM Workgroup chair

Change History

Version	Date	Description
Rev 50	Jul 2003	Started 01 Jul 2003 by David Grawrock 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: <ul style="list-style-type: none"> ▪ Update to section 6.2 informative, for TPM_OwnerClear. ▪ Addition 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 ▪ Addition 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_AUTHTHREAD 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 disableFullIDALogicInfo. 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.

TCG Doc Roadmap – Main Spec



TCG Main Spec Roadmap

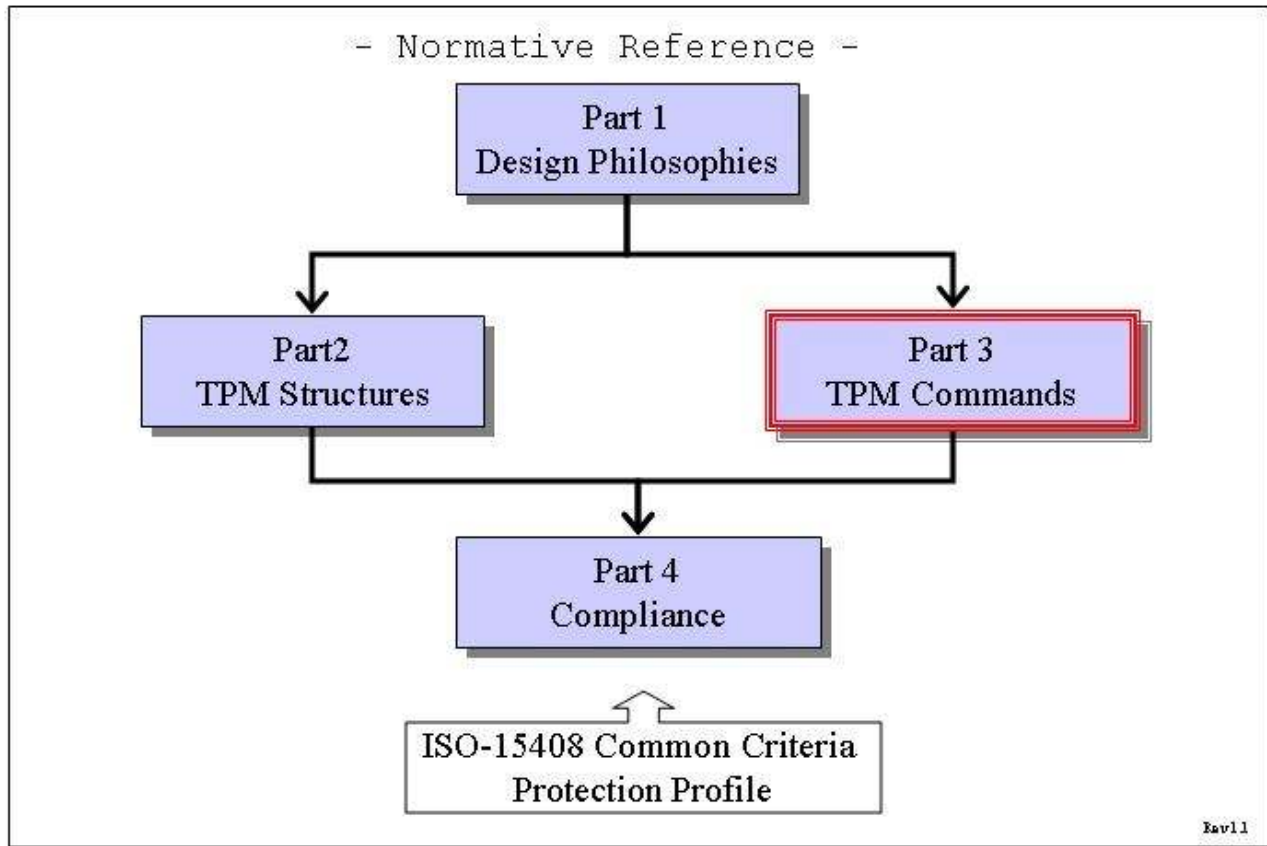


Table of Contents

1. Scope and Audience.....	2
1.1 Key words.....	2
1.2 Statement Type.....	2
2. Description and TODO	2
3. Admin Startup and State.....	2
3.1 TPM_Init.....	2
3.2 TPM_Startup.....	2
3.3 TPM_SaveState	2
4. Admin Testing.....	2
4.1 TPM_SelfTestFull.....	2
4.2 TPM_ContinueSelfTest	2
4.3 TPM_GetTestResult.....	2
5. Admin Opt-in.....	2
5.1 TPM_SetOwnerInstall	2
5.2 TPM_OwnerSetDisable.....	2
5.3 TPM_PhysicalEnable	2
5.4 TPM_PhysicalDisable	2
5.5 TPM_PhysicalSetDeactivated.....	2
5.6 TPM_SetTempDeactivated	2
5.7 TPM_SetOperatorAuth.....	2
6. Admin Ownership.....	2
6.1 TPM_TakeOwnership	2
6.2 TPM_OwnerClear.....	2
6.3 TPM_ForceClear.....	2
6.4 TPM_DisableOwnerClear	2
6.5 TPM_DisableForceClear	2
6.6 TSC_PhysicalPresence	2
6.7 TSC_ResetEstablishmentBit.....	2
7. The Capability Commands	2
7.1 TPM_GetCapability	2
7.2 TPM_SetCapability.....	2
7.3 TPM_GetCapabilityOwner	2
8. Auditing.....	2
8.1 Audit Generation	2
8.2 Effect of audit failing.....	2

8.3	TPM_GetAuditDigest	2
8.4	TPM_GetAuditDigestSigned	2
8.5	TPM_SetOrdinalAuditStatus	2
9.	Administrative Functions - Management	2
9.1	TPM_FieldUpgrade	2
9.2	TPM_SetRedirection	2
9.3	TPM_ResetLockValue	2
10.	Storage functions	2
10.1	TPM_Seal.....	2
10.2	TPM_Unseal.....	2
10.3	TPM_UnBind.....	2
10.4	TPM_CreateWrapKey	2
10.5	TPM_LoadKey2.....	2
10.6	TPM_GetPubKey	2
10.7	TPM_Sealx.....	2
11.	Migration	2
11.1	TPM_CreateMigrationBlob	2
11.2	TPM_ConvertMigrationBlob	2
11.3	TPM_AuthorizeMigrationKey	2
11.4	TPM_MigrateKey.....	2
11.5	TPM_CMK_SetRestrictions	2
11.6	TPM_CMK_ApproveMA.....	2
11.7	TPM_CMK_CreateKey.....	2
11.8	TPM_CMK_CreateTicket	2
11.9	TPM_CMK_CreateBlob.....	2
11.10	TPM_CMK_ConvertMigration	2
12.	Maintenance Functions (optional).....	2
12.1	TPM_CreateMaintenanceArchive	2
12.2	TPM_LoadMaintenanceArchive	2
12.3	TPM_KillMaintenanceFeature.....	2
12.4	TPM_LoadManuMaintPub	2
12.5	TPM_ReadManuMaintPub.....	2
13.	Cryptographic Functions	2
13.1	TPM_SHA1Start.....	2
13.2	TPM_SHA1Update.....	2
13.3	TPM_SHA1Complete.....	2
13.4	TPM_SHA1CompleteExtend.....	2

13.5	TPM_Sign.....	2
13.6	TPM_GetRandom	2
13.7	TPM_StirRandom.....	2
13.8	TPM_CertifyKey	2
13.9	TPM_CertifyKey2	2
14.	Endorsement Key Handling	2
14.1	TPM_CreateEndorsementKeyPair.....	2
14.2	TPM_CreateRevocableEK.....	2
14.3	TPM_RevokeTrust	2
14.4	TPM_ReadPubek	2
14.5	TPM_OwnerReadInternalPub.....	2
15.	Identity Creation and Activation	2
15.1	TPM_MakeIdentity	2
15.2	TPM_ActivateIdentity	2
16.	Integrity Collection and Reporting.....	2
16.1	TPM_Extend.....	2
16.2	TPM_PCRRead.....	2
16.3	TPM_Quote	2
16.4	TPM_PCR_Reset.....	2
16.5	TPM_Quote2.....	2
17.	Changing AuthData.....	2
17.1	TPM_ChangeAuth.....	2
17.2	TPM_ChangeAuthOwner	2
18.	Authorization Sessions	2
18.1	TPM_OIAP	2
18.1.1	Actions to validate an OIAP session.....	2
18.2	TPM_O SAP	2
18.2.1	Actions to validate an OSAP session.....	2
18.3	TPM_DSAP	2
18.4	TPM_SetOwnerPointer	2
19.	Delegation Commands	2
19.1	TPM_Delegate_Manage	2
19.2	TPM_Delegate_CreateKeyDelegation.....	2
19.3	TPM_Delegate_CreateOwnerDelegation	2
19.4	TPM_Delegate_LoadOwnerDelegation	2
19.5	TPM_Delegate_ReadTable.....	2
19.6	TPM_Delegate_UpdateVerification.....	2

19.7	TPM_Delegate_VerifyDelegation	2
20.	Non-volatile Storage	2
20.1	TPM_NV_DefineSpace	2
20.2	TPM_NV_WriteValue	2
20.3	TPM_NV_WriteValueAuth	2
20.4	TPM_NV_ReadValue	2
20.5	TPM_NV_ReadValueAuth	2
21.	Session Management	2
21.1	TPM_KeyControlOwner	2
21.2	TPM_SaveContext	2
21.3	TPM_LoadContext	2
22.	Eviction	2
22.1	TPM_FlushSpecific	2
23.	Timing Ticks	2
23.1	TPM_GetTicks	2
23.2	TPM_TickStampBlob	2
24.	Transport Sessions	2
24.1	TPM_EstablishTransport	2
24.2	TPM_ExecuteTransport	2
24.3	TPM_ReleaseTransportSigned	2
25.	Monotonic Counter	2
25.1	TPM_CreateCounter	2
25.2	TPM_IncrementCounter	2
25.3	TPM_ReadCounter	2
25.4	TPM_ReleaseCounter	2
25.5	TPM_ReleaseCounterOwner	2
26.	DAA commands	2
26.1	TPM_DAA_Join	2
26.2	TPM_DAA_Sign	2
27.	Deprecated commands	2
27.1	Key commands	2
27.1.1	TPM_EvictKey	2
27.1.2	TPM_Terminate_Handle	2
27.2	Context management	2
27.2.1	TPM_SaveKeyContext	2
27.2.2	TPM_LoadKeyContext	2
27.2.3	TPM_SaveAuthContext	2

27.2.4	TPM_LoadAuthContext.....	2
27.3	DIR commands.....	2
27.3.1	TPM_DirWriteAuth	2
27.3.2	TPM_DirRead	2
27.4	Change Auth	2
27.4.1	TPM_ChangeAuthAsymStart.....	2
27.4.2	TPM_ChangeAuthAsymFinish.....	2
27.5	TPM_Reset	2
27.6	TPM_OwnerReadPubek	2
27.7	TPM_DisablePubekRead.....	2
27.8	TPM_LoadKey.....	2
28.	Deleted Commands	2
28.1	TPM_GetCapabilitySigned.....	2
28.2	TPM_GetOrdinalAuditStatus.....	2
28.3	TPM_CertifySelfTest	2

End of Introduction do not delete

1 **1. Scope and Audience**

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

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

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

9 **1.1 Key words**

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

13 **1.2 Statement Type**

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

22 For example:

23 **Start of informative comment:**

24 This is the first paragraph of 1–n paragraphs containing text of the kind informative
25 comment ...

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

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

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

30 **End of informative comment.**

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

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

35 **2. Description and TODO**

36 This document is to show the changes necessary to create the 1.2 version of the TCG
37 specification. Some of the sections are brand new text; some are rewritten sections of the
38 1.1 version. Upon approval of the 1.2 changes, there will be a merging of the 1.1 and 1.2
39 versions to create a single 1.2 document.

40 **3. Admin Startup and State**

41 **Start of informative comment:**

42 This section is the commands that start a TPM.

43 **End of informative comment.**

44 **3.1 TPM_Init**

45 **Start of informative comment:**

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

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

52 **End of informative comment.**

53 **Definition**

54 `TPM_Init();`
55

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

58 **Parameters**

59 None

60 **Description**

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

73 **Actions**

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

75 **3.2 TPM_Startup**76 **Start of informative comment:**

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

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

84 The TPM can startup in three different modes:

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

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

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

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

94 **End of informative comment.**95 **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_Startup
4	2	2S	2	TPM_STARTUP_TYPE	startupType	Type of startup that is occurring

96 **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_Startup

97 **Description**

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

99 1. If the TPM is in failure mode

100 a. TPM_STANY_FLAGS -> postInitialize is still set to FALSE

101 b. The TPM returns TPM_FAILEDSELFTEST

102 **Actions**

103 1. If TPM_STANY_FLAGS -> postInitialise is FALSE,

104 a. Then the TPM MUST return TPM_INVALID_POSTINIT, and exit this capability

105 2. If stType = TPM_ST_CLEAR

106 a. Ensure that sessions associated with resources TPM_RT_CONTEXT, TPM_RT_AUTH,
107 TPM_RT_DAA_TPM, and TPM_RT_TRANS are invalidated

108 b. Reset TPM_STCLEAR_DATA -> PCR[] values to each correct default value

109 i. pcrReset is FALSE, set to 0x00..00

110 ii. pcrReset is TRUE, set to 0xFF..FF

111 c. Set the following TPM_STCLEAR_FLAGS to their default state

112 i. PhysicalPresence

113 ii. PhysicalPresenceLock

114 iii. disableForceClear

115 d. The TPM MAY initialize auditDigest to all zeros

116 i. If not initialized to all zeros, the TPM SHALL ensure that auditDigest contains a
117 valid value.

118 ii. If initialization fails, the TPM SHALL set auditDigest to all zeros and SHALL set
119 the internal TPM state so that the TPM returns TPM_FAILEDSELFTEST to all
120 subsequent commands.

121 e. The TPM SHALL set TPM_STCLEAR_FLAGS -> deactivated to the same state as
122 TPM_PERMANENT_FLAGS -> deactivated

123 f. The TPM MUST set the TPM_STANY_DATA fields to:

124 i. TPM_STANY_DATA->contextNonceSession is set to all zeros

125 ii. TPM_STANY_DATA->contextCount is set to 0

126 iii. TPM_STANY_DATA->contextList is set to 0

127 g. The TPM MUST set TPM_STCLEAR_DATA fields to:

128 i. Invalidate contextNonceKey

129 ii. countID to zero

130 iii. ownerReference to TPM_KH_OWNER

131 h. The TPM MUST set the following TPM_STCLEAR_FLAGS to

132 i. bGlobalLock to FALSE

133 i. Determine which keys should remain in the TPM

134 i. For each key that has a valid preserved value in the TPM

- 135 (1) if parentPCRStatus is TRUE then call TPM_FlushSpecific(keyHandle)
136 (2) if isVolatile is TRUE then call TPM_FlushSpecific(keyHandle)
137 ii. Keys under control of the OwnerEvict flag MUST stay resident in the TPM
138 3. If stType = TPM_ST_STATE
139 a. If the TPM has no state to restore, the TPM MUST set the internal state such that it
140 returns TPM_FAILEDSELFTEST to all subsequent commands.
141 b. The TPM MAY determine for each session type (authorization, transport, DAA, ...) to
142 release or maintain the session information. The TPM reports how it manages sessions
143 in the TPM_GetCapability command.
144 c. The TPM SHALL take all necessary actions to ensure that all PCRs contain valid
145 preserved values. If the TPM is unable to successfully complete these actions, it SHALL
146 enter the TPM failure mode.
147 i. For resettable PCR the TPM MUST set the value of TPM_STCLEAR_DATA ->
148 PCR[] to the resettable PCR default value. The TPM MUST NOT restore a resettable
149 PCR to a preserved value
150 d. The TPM MAY initialize auditDigest to all zeros.
151 i. Otherwise, the TPM SHALL take all actions necessary to ensure that auditDigest
152 contains a valid value. If the TPM is unable to successfully complete these
153 actions, the TPM SHALL initialize auditDigest to all zeros and SHALL set the
154 internal state such that the TPM returns TPM_FAILEDSELFTEST to all
155 subsequent commands.
156 e. The TPM MUST restore the following flags to their preserved states:
157 i. All values in TPM_STCLEAR_FLAGS
158 ii. All values in TPM_STCLEAR_DATA
159 f. The TPM MUST restore all keys that have a valid preserved value.
160 g. The TPM resumes normal operation. If the TPM is unable to resume normal
161 operation, it SHALL enter the TPM failure mode.
162 4. If stType = TPM_ST_DEACTIVATED
163 a. Invalidate sessions
164 i. Ensure that all resources associated with saved and active sessions are
165 invalidated
166 b. Set the TPM_STCLEAR_FLAGS to their default state.
167 c. Set TPM_STCLEAR_FLAGS -> deactivated to TRUE
168 5. The TPM MUST ensure that state associated with TPM_SaveState is invalidated
169 6. The TPM MUST set TPM_STANY_FLAGS -> postInitialise to FALSE

170 3.3 TPM_SaveState

171 Start of informative comment:

172 This warns a TPM to save some state information.

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

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

177 End of informative comment.

178 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.

179 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.

180 Description

- 181 1. Preserved values MUST be non-volatile.
- 182 2. If data is never stored in a volatile medium, that data MAY be used as preserved data. In
183 such cases, no explicit action may be required to preserve that data.
- 184 3. If an explicit action is required to preserve data, it MUST be possible for the TPM to
185 determine whether preserved data is valid.
- 186 4. If a parameter mirrored by any preserved value is altered, all preserved values MUST be
187 declared invalid.
- 188 5. The TPM MAY declare all preserved values invalid in response to any command other
189 than TPM_Init.

190 Actions

- 191 1. Store TPM_STCLEAR_DATA -> PCR contents except for
 - 192 a. If the PCR attribute pcrReset is TRUE
 - 193 b. Any platform identified debug PCR

- 194 2. The auditDigest MUST be handled according to the audit requirements as reported by
195 TPM_GetCapability.
- 196 a. If the ordinalAuditStatus is TRUE for the TPM_SaveState ordinal and the auditDigest
197 is being stored in the saved state, the saved auditDigest MUST include the
198 TPM_SaveState input parameters and MUST NOT include the output parameters.
- 199 3. All values in TPM_STCLEAR_DATA MUST be preserved.
- 200 4. All values in TPM_STCLEAR_FLAGS MUST be preserved.
- 201 5. The contents of any key that is currently loaded SHOULD be preserved if the key's
202 parentPCRStatus indicator is FALSE and its isVolatile indicator is FALSE.
- 203 6. The contents of any key that has TPM_KEY_CONTROL_OWNER_EVICT set MUST be
204 preserved
- 205 7. The contents of any key that is currently loaded MAY be preserved.
- 206 8. The contents of sessions (authorization, transport, DAA, etc.) MAY be preserved as
207 reported by TPM_GetCapability.

208 **4. Admin Testing**

209 **4.1 TPM_SelfTestFull**

210 **Start of informative comment:**

211 TPM_SelfTestFull tests all of the TPM capabilities.

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

214 **End of informative comment.**

215 **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

216 **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

217 **Actions**

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

225 **4.2 TPM_ContinueSelfTest**226 **Start of informative comment:**

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

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

231 **End of informative comment.**232 **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

233 **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

234 **Description**

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

237 a. The TPM MAY return TPM_NEEDS_SELFTEST

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

240 ii. The TPM does not execute C1.

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

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

245 b. The TPM MAY return TPM_DOING_SELFTEST

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

248 ii. The TPM does not execute C1.

- 249 iii. The caller **MUST** wait for the actions of TPM_ContinueSelfTest to complete before
250 reissuing the command C1.
- 251 c. The TPM **MAY** return TPM_SUCCESS or an error code associated with C1.
- 252 i. This indicates that the TPM has completed the actions of TPM_ContinueSelfTest
253 and has completed the command C1.
- 254 ii. The error code **MAY** be TPM_FAILEDSELFTEST.

255 **Actions**

- 256 1. If TPM_PERMANENT_FLAGS -> FIPS is TRUE or TPM_PERMANENT_FLAGS -> TPMpost
257 is TRUE
 - 258 a. The TPM **MUST** run all self-tests
- 259 2. Else
 - 260 a. The TPM **MUST** complete all self-tests that are outstanding
 - 261 i. Instead of completing all outstanding self-tests the TPM **MAY** run all self-tests
- 262 3. The TPM either
 - 263 a. **MAY** immediately return TPM_SUCCESS
 - 264 i. When TPM_ContinueSelfTest finishes execution, it **MUST NOT** respond to the
265 caller with a return code.
 - 266 b. **MAY** complete the self-test and then return TPM_SUCCESS or
267 TPM_FAILEDSELFTEST.

268 **4.3 TPM_GetTestResult**269 **Start of informative comment:**

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

274 **End of informative comment.**275 **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_GetTestResult

276 **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_GetTestResult
4	4	3S	4	UINT32	outDataSize	The size of the outData area
5	<>	4S	<>	BYTE[]	outData	The outData this is manufacturer specific

277 **Description**

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

280 **Actions**

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

284 **5. Admin Opt-in**

285 **5.1 TPM_SetOwnerInstall**

286 **Start of informative comment:**

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

289 **End of informative comment.**

290 **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.

291 **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

292 **Action**

293 1. If the TPM has a current owner, this command immediately returns with
294 TPM_SUCCESS.

295 2. The TPM validates the assertion of physical presence. The TPM then sets the value of
296 TPM_PERMANENT_FLAGS -> ownership to the value in state.

297 **5.2 TPM_OwnerSetDisable**298 **Start of informative comment:**

299 The TPM owner sets the PERMANENT disable flag

300 **End of informative comment.**301 **Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerSetDisable
4	1	2S	1	BOOL	disableState	Value for disable state – enable if TRUE
5	4			TPM_AUTHHANDLE	authHandle	The authorization 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.

302 **Outgoing Operands and Sizes**

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

303 **Action**

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

308 **5.3 TPM_PhysicalEnable**

309 **Start of informative comment:**

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

311 **End of informative comment.**

312 **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_PhysicalEnable

313 **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_PhysicalEnable

314 **Action**

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

317 **5.4 TPM_PhysicalDisable**318 **Start of informative comment:**

319 Sets the PERMANENT disable flag to TRUE using physical presence as authorization

320 **End of informative comment.**321 **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

322 **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

323 **Action**

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

326 **5.5 TPM_PhysicalSetDeactivated**

327 **Start of informative comment:**

328 Enables the TPM using physical presence as authorization.

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

330 **End of informative comment.**

331 **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_PhysicalSetDeactivated
4	1	2S	1	BOOL	state	State to which deactivated flag is to be set.

332 **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_PhysicalSetDeactivated

333 **Action**

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

337 **5.6 TPM_SetTempDeactivated**338 **Start of informative comment:**

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

341 This command requires operator authentication. The operator can provide the
342 authentication by either the assertion of physical presence or presenting the operator
343 AuthData value.

344 **End of informative comment.**345 **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

346 **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.

347 **Action**

348 1. If tag = TPM_TAG_REQ_AUTH1_COMMAND

349 a. If TPM_PERMANENT_FLAGS -> operator is FALSE return TPM_NOOPERATOR

350 b. Validate command and parameters using operatorAuth, on error return
351 TPM_AUTHFAIL

352 2. Else

- 353 a. If physical presence is not asserted the TPM MUST return TPM_BAD_PRESENCE
- 354 3. The TPM SHALL set the TPM_STCLEAR_FLAGS.deactivated flag to the value TRUE.

355 **5.7 TPM_SetOperatorAuth**356 **Start of informative comment:**

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

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

362 **End of informative comment.**363 **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

364 **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

365 **Action**

- 366 1. If physical presence is not asserted the TPM MUST return TPM_BAD_PRESENCE
- 367 2. The TPM SHALL set the TPM_PERMANENT_DATA -> operatorAuth
- 368 3. The TPM SHALL set TPM_PERMANENT_FLAGS -> operator to TRUE

369 **6. Admin Ownership**

370 **6.1 TPM_TakeOwnership**

371 **Start of informative comment:**

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

373 **End of informative comment.**

374 **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	encSrkJAuthSize	The size of the encSrkJAuth field
8	<>	6S	<>	BYTE[]	encSrkJAuth	The SRK AuthData encrypted with PUBEK
9	<>	7S	<>	TPM_KEY	srkJParams	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

375

376 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_TakeOwnership
4	<>	3S	<>	TPM_KEY	srkPub	Structure containing all parameters of new SRK. srkPub.encData is set to 0. 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

377 **Description**

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

380 **Actions**

- 381 1. If TPM_PERMANENT_DATA -> ownerAuth is valid return TPM_OWNER_SET
- 382 2. If TPM_PERMANENT_FLAGS -> ownership is FALSE return TPM_INSTALL_DISABLED
- 383 3. If TPM_PERMANENT_DATA -> endorsementKey is invalid return
384 TPM_NO_ENDORSEMENT
- 385 4. Verify that authHandle is of type OIAP on error return TPM_AUTHFAIL
- 386 5. If protocolID is not TPM_PID_OWNER, the TPM MAY return TPM_BAD_PARAMETER
- 387 6. Create A1 a TPM_SECRET by decrypting encOwnerAuth using PRIVEK as the key
388 a. This requires that A1 was encrypted using the PUBEK
389 b. Validate that A1 is a length of 20 bytes, on error return TPM_BAD_KEY_PROPERTY
- 390 7. Validate the command and parameters using A1 and ownerAuth, on error return
391 TPM_AUTHFAIL
- 392 8. Validate srkParams
393 a. If srkParams -> keyUsage is not TPM_KEY_STORAGE return
394 TPM_INVALID_KEYUSAGE
395 b. If srkParams -> migratable is TRUE return TPM_INVALID_KEYUSAGE
396 c. If srkParams -> algorithmParms -> algorithmID is NOT TPM_ALG_RSA return
397 TPM_BAD_KEY_PROPERTY
398 d. If srkParams -> algorithmParms -> encScheme is NOT
399 TPM_ES_RSAESOAEP_SHA1_MGF1 return TPM_BAD_KEY_PROPERTY

- 400 e. If srkParams -> algorithmParms -> sigScheme is NOT TPM_SS_NONE return
- 401 TPM_BAD_KEY_PROPERTY
- 402 f. If srkParams -> algorithmParms -> parms -> keyLength MUST be greater than or
- 403 equal to 2048, on error return TPM_BAD_KEY_PROPERTY
- 404 g. If TPM_PERMANENT_FLAGS -> FIPS is TRUE
- 405 i. If srkParams -> authDataUsage specifies TPM_AUTH_NEVER return
- 406 TPM_NOTFIPS
- 407 9. Generate K1 according to the srkParams, on error return TPM_BAD_KEY_PROPERTY
- 408 a. This includes copying PCRInfo from srkParams to K1
- 409 10. Create A2 a TPM_SECRET by decrypting encSrkJAuth using the PRIVEK
- 410 a. This requires A2 to be encrypted using the PUBEK
- 411 b. Validate that A2 is a length of 20 bytes, on error return TPM_BAD_KEY_PROPERTY
- 412 c. Store A2 in K1 -> usageAuth
- 413 11. Store K1 in TPM_PERMANENT_DATA -> srk
- 414 12. Store A1 in TPM_PERMANENT_DATA -> ownerAuth
- 415 13. Create TPM_PERMANENT_DATA -> contextKey according to the rules for the algorithm
- 416 in use by the TPM to save context blobs
- 417 14. Create TPM_PERMANENT_DATA -> delegateKey according to the rules for the algorithm
- 418 in use by the TPM to save delegate blobs
- 419 15. Create TPM_PERMANENT_DATA -> tpmProof by using the TPM RNG
- 420 16. Export TPM_PERMANENT_DATA -> srk as srkJPub
- 421 17. Set TPM_PERMANENT_FLAGS -> readPubek to FALSE
- 422 18. Calculate resAuth using the newly established TPM_PERMANENT_DATA -> ownerAuth

423 **6.2 TPM_OwnerClear**424 **Start of informative comment:**

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

428 **End of informative comment.**429 **Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_OwnerClear
4	4			TPM_AUTHHANDLE	authHandle	The authorization 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.

430 **Outgoing Operands and Sizes**

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

431 **Actions**

- 432 1. Verify that the TPM Owner authorizes the command and all of the input, on error return
 433 TPM_AUTHFAIL.
- 434 2. If TPM_PERMANENT_FLAGS -> disableOwnerClear is TRUE then return
 435 TPM_CLEAR_DISABLED.
- 436 3. Unload all loaded keys.

- 437 a. If TPM_PERMANENT_FLAGS -> FIPS is TRUE, the memory locations containing
438 secret or private keys MUST be set to all zeros.
- 439 4. The TPM MUST NOT modify the following TPM_PERMANENT_DATA items
- 440 a. endorsementKey
 - 441 b. revMajor
 - 442 c. revMinor
 - 443 d. manuMaintPub
 - 444 e. auditMonotonicCounter
 - 445 f. monotonicCounter
 - 446 g. pcrAttrib
 - 447 h. rngState
 - 448 i. EKReset
 - 449 j. maxNVBufSize
 - 450 k. lastFamilyID
 - 451 l. tpmDAASeed
 - 452 m. authDIR[0]
 - 453 n. daaProof
 - 454 o. daaBlobKey
- 455 5. The TPM MUST invalidate the following TPM_PERMANENT_DATA items and any internal
456 resources associated with these items
- 457 a. ownerAuth
 - 458 b. srk
 - 459 c. delegateKey
 - 460 d. delegateTable
 - 461 e. contextKey
 - 462 f. tpmProof
 - 463 g. operatorAuth
- 464 6. The TPM MUST reset to manufacturing defaults the following TPM_PERMANENT_DATA
465 items
- 466 a. noOwnerNVWrite MUST be set to 0
 - 467 b. ordinalAuditStatus
 - 468 c. restrictDelegate
- 469 7. The TPM MUST invalidate or reset all fields of TPM_STANY_DATA
- 470 a. Nonces SHALL be reset
 - 471 b. Lists (e.g. contextList) SHALL be invalidated

- 472 8. The TPM MUST invalidate or reset all fields of TPM_STCLEAR_DATA except the PCR's
473 a. Nonces SHALL be reset
474 b. Lists (e.g. contextList) SHALL be invalidated
475 c. deferredPhysicalPresence MUST be set to 0
476 9. The TPM MUST set the following TPM_PERMANENT_FLAGS to their default values
477 a. disable
478 b. deactivated
479 c. readPubek
480 d. disableOwnerClear
481 e. disableFullDALogicInfo
482 10. The TPM MUST set the following TPM_PERMANENT_FLAGS
483 a. ownership to TRUE
484 b. operator to FALSE
485 c. maintenanceDone to FALSE
486 d. allowMaintenance to TRUE
487 11. The TPM releases all TPM_PERMANENT_DATA -> monotonicCounter settings
488 a. This includes invalidating all currently allocated counters. The result will be no
489 currently allocated counters and the new owner will need to allocate counters. The
490 actual count value will continue to increase.
491 12. The TPM MUST deallocate all defined NV storage areas where
492 a. TPM_NV_PER_OWNERWRITE is TRUE if nvIndex does not have the "D" bit set
493 b. TPM_NV_PER_OWNERREAD is TRUE if nvIndex does not have the "D" bit set
494 c. The TPM MUST NOT deallocate any other currently defined NV storage areas.
495 13. The TPM MUST invalidate all familyTable entries
496 14. The TPM MUST terminate all sessions, active or saved.

497 **6.3 TPM_ForceClear**

498 **Start of informative comment:**

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

502 **End of informative comment.**

503 **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

504 **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

505 **Actions**

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

511 **6.4 TPM_DisableOwnerClear**512 **Start of informative comment:**

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

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

518 **End of informative comment.**519 **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.

520 **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.

521 **Actions**

- 522 1. The TPM verifies that the authHandle properly authorizes the owner.
- 523 2. The TPM sets the TPM_PERMANENT_FLAGS -> disableOwnerClear flag to TRUE.

524 3. When this flag is TRUE the only mechanism that can clear the TPM is the
525 TPM_ForceClear command. The TPM_ForceClear command requires physical access to
526 the TPM to execute.

527 **6.5 TPM_DisableForceClear**

528 Start of informative comment:

529 The TPM_DisableForceClear command disables the execution of the TPM_ForceClear
530 command until the next startup cycle. Once this command is executed, the TPM_ForceClear
531 is disabled until another startup cycle is run.

532 End of informative comment.

533 **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

534 **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

535 **Actions**536 1. The TPM sets the TPM_STCLEAR_FLAGS.disableForceClear flag in the TPM that disables
537 the execution of the TPM_ForceClear command.

538 **6.6 TSC_PhysicalPresence**

539 **Start of informative comment:**

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

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

547 The physicalPresenceHwEnable and physicalPresenceCmdEnable indicate the ability for
548 either SW or HW to indicate physical presence. These flags can be reset until the
549 physicalPresenceLifetimeLock is set. The platform manufacturer should set these flags to
550 indicate the capabilities of the platform the TPM is bound to.

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

554 **End of informative comment.**

555 **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.

556 **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.

557 **Actions**

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

560 a. Define A1 to be the lifetime settings: TPM_PHYSICAL_PRESENCE_LIFETIME_LOCK,
561 TPM_PHYSICAL_PRESENCE_HW_ENABLE, TPM_PHYSICAL_PRESENCE_CMD_ENABLE,
562 TPM_PHYSICAL_PRESENCE_HW_DISABLE, and
563 TPM_PHYSICAL_PRESENCE_CMD_DISABLE

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

566 Lifetime lock settings

567 2. If any A1 setting is present

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

570 b. If any A2 setting is present return TPM_BAD_PARAMETER

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

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

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

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

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

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

585 i. If physicalPresence -> TPM_PHYSICAL_PRESENCE_LIFETIME_LOCK is TRUE

586 i. Set TPM_PERMANENT_FLAGS -> physicalPresenceLifetimeLock to TRUE

587 j. Return TPM_SUCCESS

588 SW physical presence assertion

589 3. If any A2 setting is present

590 a. If any A1 setting is present return TPM_BAD_PARAMETER

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

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

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

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

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

- 602 f. If physicalPresence -> TPM_PHYSICAL_PRESENCE_LOCK is TRUE
- 603 i. Set TPM_STCLEAR_FLAGS -> physicalPresence to FALSE
- 604 ii. Set TPM_STCLEAR_FLAGS -> physicalPresenceLock to TRUE
- 605 iii. Return TPM_SUCCESS
- 606 g. If physicalPresence -> TPM_PHYSICAL_PRESENCE_PRESENT is TRUE
- 607 i. Set TPM_STCLEAR_FLAGS -> physicalPresence to TRUE
- 608 h. If physicalPresence -> TPM_PHYSICAL_PRESENCE_NOTPRESENT is TRUE
- 609 i. Set TPM_STCLEAR_FLAGS -> physicalPresence to FALSE
- 610 i. Return TPM_SUCCESS
- 611 4. Else // There were no A1 or A2 parameters set
- 612 a. Return TPM_BAD_PARAMETER

613 **6.7 TSC_ResetEstablishmentBit**614 **Start of informative comment:**

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

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

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

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

629 **End of informative comment.**630 **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_ResetEstablishmentBit

631 **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_ResetEstablishmentBit

632 **Actions**

- 633 1. Validate the assertion of locality 3 or locality 4
- 634 2. Set TPM_PERMANENT_FLAGS -> tpmEstablished to FALSE
- 635 3. Return TPM_SUCCESS

636 7. The Capability Commands

637 **Start of informative comment:**

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

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

645 In failure mode, the TPM returns a limited set of information that includes the TPM
646 manufacturer and version.

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

651 **End of informative comment.**

652 1. The standard information provided in TPM_GetCapability MUST NOT provide unique
653 information

654 a. The TPM has no control of information placed into areas on the TPM like the NV store
655 that is reported by the TPM. Configuration information for these areas could conceivably
656 be unique

657 **7.1 TPM_GetCapability**658 **Start of informative comment:**

659 This command returns current information regarding the TPM.

660 The limitation on what can be returned in failure mode restricts the information a
661 manufacturer may return when capArea indicates TPM_CAP_MFR.662 **End of informative comment.**663 **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

664 **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

665

666 **Actions**

- 667 1. The TPM validates the capArea and subCap indicators. If the information is available,
668 the TPM creates the response field and fills in the actual information.
- 669 2. The structure document contains the list of caparea and subCap values
- 670 3. If the TPM is in failure mode or limited operation mode, the TPM MUST return
- 671 a. TPM_CAP_VERSION
- 672 b. TPM_CAP_VERSION_VAL
- 673 c. TPM_CAP_MFR
- 674 d. TPM_CAP_PROPERTY -> TPM_CAP_PROP_MANUFACTURER
- 675 e. TPM_CAP_PROPERTY -> TPM_CAP_PROP_DURATION
- 676 f. TPM_CAP_PROPERTY -> TPM_CAP_PROP_TIS_TIMEOUT
- 677 g. The TPM MAY return any other capability.

678 **7.2 TPM_SetCapability**679 **Start of informative comment:**

680 This command sets values in the TPM.

681 A setValue that is inconsistent with the capArea and subCap is considered a bad
682 parameter.683 **End of informative comment.**684 **Incoming Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	ordinal: TPM_ORD_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.

685 **Outgoing Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	ordinal: TPM_ORD_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.

686 **Actions**

- 687 1. If tag = TPM_TAG_RQU_AUTH1_COMMAND, validate the command and parameters
688 using ownerAuth, return TPM_AUTHFAIL on error
- 689 2. The TPM validates the capArea and subCap indicators, including the ability to set value
690 based on any set restrictions
- 691 3. If the capArea and subCap indicators conform with one of the entries in the structure
692 TPM_CAPABILITY_AREA (Values for TPM_SetCapability)
- 693 a. The TPM sets the relevant flag/data to the value of setValue parameter.
- 694 4. Else
- 695 a. Return the error code TPM_BAD_PARAMETER.

696 **7.3 TPM_GetCapabilityOwner**697 **Start of informative comment:**

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

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

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

706 **End of informative comment.**707 **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_GetCapabilityOwner
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.

708

709 **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.

710

711 **Description**

712 For $31 \geq N \geq 0$

- 713 1. Bit-N of the TPM_PERMANENT_FLAGS structure is the Nth bit after the opening bracket
714 in the definition of TPM_PERMANENT_FLAGS in the version of the specification
715 indicated by the parameter “version”. The bit immediately after the opening bracket is
716 the 0th bit.
- 717 2. Bit-N of the TPM_STCLEAR_FLAGS structure is the Nth bit after the opening bracket in
718 the definition of TPM_STCLEAR_FLAGS in the version of the specification indicated by
719 the parameter “version”. The bit immediately after the opening bracket is the 0th bit.
- 720 3. Bit-N of non_volatile_flags corresponds to the Nth bit in TPM_PERMANENT_FLAGS, and
721 the lsb of non_volatile_flags corresponds to bit0 of TPM_PERMANENT_FLAGS
- 722 4. Bit-N of volatile_flags corresponds to the Nth bit in TPM_STCLEAR_FLAGS, and the lsb
723 of volatile_flags corresponds to bit0 of TPM_STCLEAR_FLAGS

724 **Actions**

- 725 1. The TPM validates that the TPM Owner authorizes the command.
- 726 2. The TPM creates the parameter non_volatile_flags by setting each bit to the same state
727 as the corresponding bit in TPM_PERMANENT_FLAGS. Bits in non_volatile_flags for
728 which there is no corresponding bit in TPM_PERMANENT_FLAGS are set to zero.
- 729 3. The TPM creates the parameter volatile_flags by setting each bit to the same state as the
730 corresponding bit in TPM_STCLEAR_FLAGS. Bits in volatile_flags for which there is no
731 corresponding bit in TPM_STCLEAR_FLAGS are set to zero.
- 732 4. The TPM generates the parameter “version”.
- 733 5. The TPM returns non_volatile_flags, volatile_flags and version to the caller.

734 **8. Auditing**735 **8.1 Audit Generation**736 **Start of informative comment:**

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

739 The TPM maintains an extended value for all audited operations.

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

742 **End of informative comment.**743 **Description**

744 1. The TPM extends the audit digest whenever the ordinalAuditStatus is TRUE for the
745 ordinal about to be executed. The only exception is if the ordinal about to be executed is
746 TPM_SetOrdinalAuditStatus. In that case, output parameter auditing is performed if the
747 ordinalAuditStatus resulting from command execution is TRUE.

748 2. If the command is malformed

749 a. If the ordinal is unknown, unimplemented, or cannot be determined, no auditing is
750 performed.

751 b. If the ordinal is known and audited, but the “above the line” parameters are
752 malformed and the input parameter digest cannot be determined, use an input digest of
753 all zeros.

754 i. Use an output digest of the return code and ordinal.

755 c. If the ordinal is known and audited, the “above the line” parameters are determined,
756 but the “below the line” parameters are malformed, use an input digest of the “above the
757 line” parameters.

758 i. Use an output digest of the return code and ordinal.

759 d. Malformed in this context means that, when breaking up a command into its
760 parameters, there are too few or too many bytes in the command stream.

761 e. Breaking up a command in this context means only the parsing required to extract
762 the parameters.

763 i. E.g., for parameter set comprising a UINT32 size and a BYTE[] array, the BYTE[]
764 array should not be further parsed.

765 f. If the ordinal returns an error because the TPM is deactivated, disabled, or has no
766 owner, auditing is performed.

767 g. If the ordinal returns an error because the input tag is invalid for the command,
768 auditing is performed.

769 **Actions**

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

- 771 1. Map V1 to TPM_STANY_DATA
- 772 2. Map P1 to TPM_PERMANENT_DATA
- 773 3. If V1 -> auditDigest is all zeros
 - 774 a. Increment P1 -> auditMonotonicCounter by 1
- 775 4. Create A1 a TPM_AUDIT_EVENT_IN structure
 - 776 a. Set A1 -> inputParms to the digest of the input parameters from the command
 - 777 i. Digest value according to the HMAC digest rules of the "above the line"
 - 778 parameters (i.e. the first HMAC digest calculation).
 - 779 b. Set A1 -> auditCount to P1 -> auditMonotonicCounter
 - 780 c. Set V1 -> auditDigest to SHA-1 (V1 -> auditDigest || A1)
- 781 5. Execute command
 - 782 a. Execution implies the performance of the listed actions for the ordinal.
- 783 6. Create A2 a TPM_AUDIT_EVENT_OUT structure
 - 784 a. Set A2 -> outputParms to the digest of the output parameters from the command
 - 785 i. Digest value according to the HMAC digest rules of the "above the line"
 - 786 parameters (i.e. the first HMAC digest calculation).
 - 787 b. Set A2 -> auditCount to P1 -> auditMonotonicCounter
 - 788 c. Set V1 -> auditDigest to SHA-1 (V1 -> auditDigest || A2)

789 **8.2 Effect of audit failing**790 **Start of informative comment:**

791 The TPM audit process could have an internal error when attempting to audit a command.

792 With one return parameter, The TPM is unable to return both the audit failure and the
793 command success or failure results. To indicate the audit failure, the TPM will return one of
794 two error codes: TPM_AUDITFAIL_SUCCESSFUL (if the command completed successfully)
795 or TPM_AUDITFAIL_UNSUCCESSFUL (if the command completed unsuccessfully).

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

797 **End of informative comment.**

- 798 1. When, in performing the audit process, the TPM has an internal failure (unable to write,
799 SHA-1 failure etc.) the TPM MUST set the internal TPM state such that the TPM returns
800 the TPM_FAILEDSELFTEST error on subsequent attempts to execute a command
- 801 2. The return code for the command uses the following rules
- 802 a. Command result success, Audit success -> return TPM_SUCCESS
 - 803 b. Command result failure, Audit success -> return command result failure
 - 804 c. Command result success, Audit failure -> return TPM_AUDITFAIL_SUCCESSFUL
 - 805 d. Command result failure, Audit failure -> return TPM_AUDITFAIL_UNSUCCESSFUL
- 806 3. If the TPM is permanently nonrecoverable after an audit failure, then the TPM MUST
807 always return TPM_FAILEDSELFTEST for every command other than
808 TPM_GetTestResult. This state must persist regardless of power cycling, the execution of
809 TPM_Init or any other actions.

810 **8.3 TPM_GetAuditDigest**

811 **Start of informative comment:**

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

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

817 **End of informative comment.**

818 **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

819 **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.

820 **Description**

821 1. This command is never audited.

822 **Actions**

823 1. The TPM sets auditDigest to TPM_STANY_DATA -> auditDigest

824 2. The TPM sets counterValue to TPM_PERMANENT_DATA -> auditMonotonicCounter

825 3. The TPM creates an ordered list of audited ordinals. The list starts at startOrdinal listing
826 each ordinal that is audited.

827 a. If startOrdinal is 0 then the first ordinal that could be audited would be TPM_OIAP
828 (ordinal 0x0000000A)

- 829 b. The next ordinal would be TPM_OSAP (ordinal 0x0000000B)
- 830 4. If the ordered list does not fit in the output buffer the TPM sets more to TRUE
- 831 5. Return TPM_STANY_DATA -> auditDigest as auditDigest

832 8.4 TPM_GetAuditDigestSigned

833 Start of informative comment:

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

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

838 Note to future architects

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

848 End of informative comment.

849 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.

850 **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.

851 **Actions**

- 852 1. Validate the AuthData and parameters using keyAuth, return TPM_AUTHFAIL on error
- 853 2. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING, TPM_KEY_IDENTITY or
- 854 TPM_KEY_LEGACY, if not return TPM_INVALID_KEYUSAGE
- 855 3. The TPM validates that the key pointed to by keyHandle has a signature scheme of
- 856 TPM_SS_RSASSAPKCS1v15_SHA1 or TPM_SS_RSASSAPKCS1v15_INFO, return
- 857 TPM_INVALID_KEYUSAGE on error
- 858 4. Create D1 a TPM_SIGN_INFO structure and set the structure defaults
- 859 a. Set D1 -> fixed to “ADIG”
- 860 b. Set D1 -> replay to antiReplay
- 861 c. Create D3 a list of all audited ordinals as defined in the TPM_GetAuditDigest
- 862 UINT32[] ordList outgoing parameter
- 863 d. Create D4 the SHA-1 of D3
- 864 e. Set auditDigest to TPM_STANY_DATA -> auditDigest
- 865 f. Set counterValue to TPM_PERMANENT_DATA -> auditMonotonicCounter
- 866 g. Create D2 the concatenation of auditDigest || counterValue || D4
- 867 h. Set D1 -> data to D2
- 868 i. Create a digital signature of the SHA-1 of D1 by using the signature scheme for
- 869 keyHandle
- 870 j. Set ordinalDigest to D4
- 871 5. If closeAudit == TRUE

- 872 a. If keyHandle->keyUsage is TPM_KEY_IDENTITY
- 873 i. TPM_STANY_DATA -> auditDigest MUST be set to all zeros.
- 874 b. Else
- 875 i. Return TPM_INVALID_KEYUSAGE

876 **8.5 TPM_SetOrdinalAuditStatus**877 **Start of informative comment:**

878 Set the audit flag for a given ordinal. Requires the authentication of the TPM Owner.

879 **End of informative comment.**880 **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.

881 **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.

882 **Actions**

- 883 1. Validate the AuthData to execute the command and the parameters
- 884 2. Validate that the ordinal points to a valid TPM ordinal, return TPM_BADINDEX on error
- 885 a. Valid TPM ordinal means an ordinal that the TPM implementation supports
- 886 3. Set the non-volatile flag associated with ordinalToAudit to the value in auditState

887 9. Administrative Functions - Management

888 9.1 TPM_FieldUpgrade

889 **Start of informative comment:**

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

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

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

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

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

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

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

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

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

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

922 **End of informative comment.**

923 **IDL Definition**

```
924 TPM_RESULT TPM_FieldUpgrade(  
925     [in, out] TPM_AUTH* ownerAuth,  
926     ...);
```

927 **Type**

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

930 **Parameters**

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

931 **Descriptions**

932 The patch integrity and authenticity verification mechanisms in the TPM MUST not require
933 the TPM to hold a global secret. The definition of global secret is a secret value shared by
934 more than one TPM.

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

939 The upgrade process can only change protected capabilities.

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

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

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

946 When a field upgrade occurs, it is always sufficient to put the TPM into the same state as a
947 successfully executed TPM_RevokeTrust.

948 **Actions**

949 The TPM SHALL perform the following when executing the command:

950 1. If TPM Owner is installed

951 a. Validate the command and parameters using TPM owner authentication, return
952 TPM_AUTHFAIL on error

953 2. Else

954 a. If TPM_STCLEAR_DATA -> deferredPhysicalPresence -> unownedFieldUpgrade is
955 FALSE return TPM_BAD_PRESENCE.

956 3. Validate that the upgrade information was sent by the TPME. The validation mechanism
957 MUST use a strength of function that is at least the same strength of function as a
958 digital signature performed using a 2048 bit RSA key.

959 4. Validate that the upgrade target is the appropriate TPM model and version.

960 5. Process the upgrade information and update the protected capabilities

- 961 6. Set the TPM_PERMANENT_DATA -> revMajor and TPM_PERMANENT_DATA -> revMinor
962 to the values indicated in the upgrade. The selection of the value is a manufacturer
963 option.
- 964 a. The TPM MAY validate that the upgrade major and minor revision are monotonically
965 increasing.
- 966 b. The TPM MAY allow upgrade with a major and minor revision that is less than
967 currently installed in the TPM.
- 968 7. Set the TPM_STCLEAR_FLAGS.deactivated to TRUE

969 **9.2 TPM_SetRedirection**970 **Informative comment**

971 The redirection command attaches a key to a redirection receiver.

972 When making the connection to a GPIO channel the authorization restrictions are set at
973 connection time and not for each invocation that uses the channel.974 **End of informative comments**975 **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

976 **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

977

978 **Action**

979 1. If tag == TPM_TAG_REQ_AUTH1_COMMAND

- 980 a. Validate the command and parameters using TPM Owner authentication, on error
981 return TPM_AUTHFAIL
- 982 2. if `redirCmd == TPM_REDIR_GPIO`
- 983 a. Validate that `keyHandle` points to a loaded key, return `TPM_INVALID_KEYHANDLE`
984 on error
- 985 b. Validate the key attributes specify redirection, return `TPM_BAD_TYPE` on error
- 986 c. Validate that `inputDataSize` is 4, return `TPM_BAD_PARAM_SIZE` on error
- 987 d. Validate that `inputData` points to a valid GPIO channel, return
988 `TPM_BAD_PARAMETER` on error
- 989 e. Map `C1` to the `TPM_GPIO_CONFIG_CHANNEL` structure indicated by `inputData`
- 990 f. If `C1 -> attr` specifies `TPM_GPIO_ATTR_OWNER`
- 991 i. If `tag != TPM_TAG_REQ_AUTH1_COMMAND` return `TPM_AUTHFAIL`
- 992 g. If `C1 -> attr` specifies `TPM_GPIO_ATTR_PP`
- 993 i. If `TPM_STCLEAR_FLAGS -> physicalPresence == FALSE`, then return
994 `TPM_BAD_PRESENCE`
- 995 h. Return `TPM_SUCCESS`
- 996 3. The TPM MAY support other redirection types. These types may be specified by TCG or
997 provided by the manufacturer.

998 **9.3 TPM_ResetLockValue**999 **Informative comment**

1000 Command that resets the TPM dictionary attack mitigation values

1001 This allows the TPM owner to cancel the effect of a number of successive authorization
 1002 failures. Dictionary attack mitigation is vendor specific, and the actions here are one
 1003 possible implementation. The TPM may treat an authorization failure outside the mitigation
 1004 time as a normal failure and not disable the command.

1005 If this command itself has an authorization failure, it is blocked for the remainder of the
 1006 lock out period. This prevents a dictionary attack on the owner authorization using this
 1007 command.

1008 It is understood that this command allows the TPM owner to perform a dictionary attack on
 1009 other authorization values by alternating a trial and this command. Similarly, delegating
 1010 this command allows the owner's delegate to perform a dictionary attack.

1011 **End of informative comments**1012 **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

1013 **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

1014 **Action**

- 1015 1. If TPM_STCLEAR_DATA -> disableResetLock is TRUE return TPM_AUTHFAIL
- 1016 a. The internal dictionary attack mechanism will set TPM_STCLEAR_DATA ->
1017 disableResetLock to FALSE when the timeout period expires
- 1018 2. If the command and parameters validation using ownerAuth fails
- 1019 a. Set TPM_STCLEAR_DATA -> disableResetLock to TRUE
- 1020 b. Restart the TPM dictionary attack lock out period
- 1021 c. Return TPM_AUTHFAIL
- 1022 3. Reset the internal TPM dictionary attack mitigation mechanism
- 1023 a. The mechanism is vendor specific and can include time outs, reboots, and other
1024 mitigation strategies

1025 **10. Storage functions**

1026 **10.1 TPM_Seal**

1027 **Start of informative comment:**

1028 The SEAL operation allows software to explicitly state the future “trusted” configuration that
1029 the platform must be in for the secret to be revealed. The SEAL operation also implicitly
1030 includes the relevant platform configuration (PCR-values) when the SEAL operation was
1031 performed. The SEAL operation uses the tpmProof value to BIND the blob to an individual
1032 TPM.

1033 If the UNSEAL operation succeeds, proof of the platform configuration that was in effect
1034 when the SEAL operation was performed is returned to the caller, as well as the secret data.
1035 This proof may, or may not, be of interest. If the SEALED secret is used to authenticate the
1036 platform to a third party, a caller is normally unconcerned about the state of the platform
1037 when the secret was SEALED, and the proof may be of no interest. On the other hand, if the
1038 SEALED secret is used to authenticate a third party to the platform, a caller is normally
1039 concerned about the state of the platform when the secret was SEALED. Then the proof is of
1040 interest.

1041 For example, if SEAL is used to store a secret key for a future configuration (probably to
1042 prove that the platform is a particular platform that is in a particular configuration), the
1043 only requirement is that that key can be used only when the platform is in that future
1044 configuration. Then there is no interest in the platform configuration when the secret key
1045 was SEALED. An example of this case is when SEAL is used to store a network
1046 authentication key.

1047 On the other hand, suppose an OS contains an encrypted database of users allowed to log
1048 on to the platform. The OS uses a SEALED blob to store the encryption key for the user-
1049 database. However, the nature of SEAL is that any SW stack can SEAL a blob for any other
1050 software stack. Hence, the OS can be attacked by a second OS replacing both the SEALED-
1051 blob encryption key, and the user database itself, allowing untrusted parties access to the
1052 services of the OS. To thwart such attacks, SEALED blobs include the past SW
1053 configuration. Hence, if the OS is concerned about such attacks, it may check to see
1054 whether the past configuration is one that is known to be trusted.

1055 TPM_Seal requires the encryption of one parameter (“Secret”). For the sake of uniformity
1056 with other commands that require the encryption of more than one parameter, the string
1057 used for XOR encryption is generated by concatenating a nonce (created during the OSAP
1058 session) with the session shared secret and then hashing the result.

1059 **End of informative comment.**

1060 **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_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.

1061 **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_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.

1062 **Descriptions**

1063 TPM_Seal is used to encrypt private objects that can only be decrypted using TPM_Unseal.

1064 **Actions**

- 1065 1. Validate the authorization to use the key pointed to by keyHandle
- 1066 2. If the inDataSize is 0 the TPM returns TPM_BAD_PARAMETER

- 1067 3. If the keyUsage field of the key indicated by keyHandle does not have the value
1068 TPM_KEY_STORAGE, the TPM must return the error code TPM_INVALID_KEYUSAGE.
- 1069 4. If the keyHandle points to a migratable key then the TPM MUST return the error code
1070 TPM_INVALID_KEY_USAGE.
- 1071 5. Determine the version of pcrInfo
- 1072 a. If pcrInfoSize is 0
- 1073 i. set V1 to 1
- 1074 b. Else
- 1075 i. Point X1 as TPM_PCR_INFO_LONG structure to pcrInfo
- 1076 ii. If X1 -> tag is TPM_TAG_PCR_INFO_LONG
- 1077 (1) Set V1 to 2
- 1078 iii. Else
- 1079 (1) Set V1 to 1
- 1080 6. If V1 is 1 then
- 1081 a. Create S1 a TPM_STORED_DATA structure
- 1082 7. else
- 1083 a. Create S1 a TPM_STORED_DATA12 structure
- 1084 b. Set S1 -> et to 0
- 1085 8. Set s1 -> encDataSize to 0
- 1086 9. Set s1 -> encData to all zeros
- 1087 10. Set s1 -> sealInfoSize to pcrInfoSize
- 1088 11. If pcrInfoSize is not 0 then
- 1089 a. if V1 is 1 then
- 1090 i. Validate pcrInfo as a valid TPM_PCR_INFO structure, return TPM_BADINDEX on
1091 error
- 1092 ii. Set s1 -> sealInfo -> pcrSelection to pcrInfo -> pcrSelection
- 1093 iii. Create h1 the composite hash of the PCR selected by pcrInfo -> pcrSelection
- 1094 iv. Set s1 -> sealInfo -> digestAtCreation to h1
- 1095 v. Set s1 -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease
- 1096 b. else
- 1097 i. Validate pcrInfo as a valid TPM_PCR_INFO_LONG structure, return
1098 TPM_BADINDEX on error
- 1099 ii. Set s1 -> sealInfo -> creationPCRSelection to pcrInfo -> creationPCRSelection
- 1100 iii. Set s1 -> sealInfo -> releasePCRSelection to pcrInfo -> releasePCRSelection
- 1101 iv. Set s1 -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease

- 1102 v. Set s1 -> sealInfo -> localityAtRelease to pcrInfo -> localityAtRelease
- 1103 vi. Create h2 the composite hash of the TPM_STCLEAR_DATA -> PCR selected by
- 1104 pcrInfo -> creationPCRSelection
- 1105 vii. Set s1 -> sealInfo -> digestAtCreation to h2
- 1106 viii. Set s1 -> sealInfo -> localityAtCreation to TPM_STANY_FLAGS ->
- 1107 localityModifier
- 1108 12. Create a1 by decrypting encAuth according to the ADIP indicated by authHandle.
- 1109 13. The TPM provides NO validation of a1. Well-known values (like all zeros) are valid and
- 1110 possible.
- 1111 14. Create s2 a TPM_SEALED_DATA structure
- 1112 a. Set s2 -> payload to TPM_PT_SEAL
- 1113 b. Set s2 -> tpmProof to TPM_PERMANENT_DATA -> tpmProof
- 1114 c. Create h3 the SHA-1 of s1
- 1115 d. Set s2 -> storedDigest to h3
- 1116 e. Set s2 -> authData to a1
- 1117 f. Set s2 -> dataSize to inDataSize
- 1118 g. Set s2 -> data to inData
- 1119 15. Validate that the size of s2 can be encrypted by the key pointed to by keyHandle, return
- 1120 TPM_BAD_DATASIZE on error
- 1121 16. Create s3 the encryption of s2 using the key pointed to by keyHandle
- 1122 17. Set continueAuthSession to FALSE
- 1123 18. Set s1 -> encDataSize to the size of s3
- 1124 19. Set s1 -> encData to s3
- 1125 20. Return s1 as sealedData

1126 **10.2 TPM_Unseal**1127 **Start of informative comment:**

1128 The TPM_Unseal operation will reveal TPM_Seal'ed data only if it was encrypted on this
 1129 platform and the current configuration (as defined by the named PCR contents) is the one
 1130 named as qualified to decrypt it. Internally, TPM_Unseal accepts a data blob generated by a
 1131 TPM_Seal operation. TPM_Unseal decrypts the structure internally, checks the integrity of
 1132 the resulting data, and checks that the PCR named has the value named during TPM_Seal.
 1133 Additionally, the caller must supply appropriate AuthData for blob and for the key that was
 1134 used to seal that data.

1135 If the integrity, platform configuration and authorization checks succeed, the sealed data is
 1136 returned to the caller; otherwise, an error is generated.

1137 **End of informative comment.**1138 **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.

1139 **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.

1140 **Actions**

- 1141 1. The TPM MUST validate that parentAuth authorizes the use of the key in parentHandle,
1142 on error return TPM_AUTHFAIL
- 1143 2. If the keyUsage field of the key indicated by parentHandle does not have the value
1144 TPM_KEY_STORAGE, the TPM MUST return the error code TPM_INVALID_KEYUSAGE.
- 1145 3. The TPM MUST check that the TPM_KEY_FLAGS -> Migratable flag has the value FALSE
1146 in the key indicated by parentHandle. If not, the TPM MUST return the error code
1147 TPM_INVALID_KEYUSAGE
- 1148 4. Determine the version of inData
 - 1149 a. If inData -> tag = TPM_TAG_STORED_DATA12
 - 1150 i. Set V1 to 2
 - 1151 ii. Map S2 a TPM_STORED_DATA12 structure to inData
 - 1152 b. Else If inData -> ver = 1.1
 - 1153 i. Set V1 to 1
 - 1154 ii. Map S2 a TPM_STORED_DATA structure to inData
 - 1155 c. Else
 - 1156 i. Return TPM_BAD_VERSION
- 1157 5. Create d1 by decrypting S2 -> encData using the key pointed to by parentHandle

- 1158 6. Validate d1
- 1159 a. d1 MUST be a TPM_SEALED_DATA structure
- 1160 b. d1 -> tpmProof MUST match TPM_PERMANENT_DATA -> tpmProof
- 1161 c. Set S2 -> encDataSize to 0
- 1162 d. Set S2 -> encData to all zeros
- 1163 e. Create h1 the SHA-1 of inData
- 1164 f. d1 -> storedDigest MUST match h1
- 1165 g. d1 -> payload MUST be TPM_PT_SEAL
- 1166 h. Any failure MUST return TPM_NOTSEALED_BLOB
- 1167 7. If S2 -> sealInfoSize is not 0 then
- 1168 a. If V1 is 1 then
- 1169 i. Validate that S2 -> pcrInfo is a valid TPM_PCR_INFO structure
- 1170 ii. Create h2 the composite hash of the PCR selected by S2 -> pcrInfo -> pcrSelection
- 1171 b. If V1 is 2 then
- 1172 i. Validate that S2 -> pcrInfo is a valid TPM_PCR_INFO_LONG structure
- 1173 ii. Create h2 the composite hash of the TPM_STCLEAR_DATA -> PCR selected by S2
- 1174 -> pcrInfo -> releasePCRSelection
- 1175 iii. Check that S2 -> pcrInfo -> localityAtRelease for TPM_STANY_DATA ->
- 1176 localityModifier is TRUE
- 1177 (1) For example if TPM_STANY_DATA -> localityModifier was 2 then S2 -> pcrInfo
- 1178 -> localityAtRelease -> TPM_LOC_TWO would have to be TRUE
- 1179 c. Compare h2 with S2 -> pcrInfo -> digestAtRelease, on mismatch return
- 1180 TPM_WRONGPCRVAL
- 1181 8. The TPM MUST validate authorization to use d1 by checking that the HMAC calculation
- 1182 using d1 -> authData as the shared secret matches the dataAuth. Return
- 1183 TPM_AUTHFAIL on mismatch.
- 1184 9. If V1 is 2 and S2 -> et specifies encryption (i.e. is not all zeros) then
- 1185 a. If tag is not TPM_TAG_RQU_AUTH2_COMMAND, return TPM_AUTHFAIL
- 1186 b. Verify that the authHandle session type is TPM_PID_OSAP, return TPM_BAD_MODE
- 1187 on error.
- 1188 c. If the MSB of S2 -> et is TPM_ET_XOR
- 1189 i. Use MGF1 to create string X1 of length sealedDataSize. The inputs to MGF1 are;
- 1190 authLastnonceEven, nonceOdd, "XOR", and authHandle -> sharedSecret. The
- 1191 four concatenated values form the Z value that is the seed for MFG1.
- 1192 ii. Create o1 by XOR of d1 -> data and X1
- 1193 d. Else
- 1194 i. Create o1 by encrypting d1 -> data using the algorithm indicated by inData -> et

- 1195 ii. Key is from authHandle -> sharedSecret
- 1196 iii. IV is SHA-1 of (authLastNonceEven || nonceOdd)
- 1197 e. Set continueAuthSession to FALSE
- 1198 10.else
- 1199 a. Set o1 to d1 -> data
- 1200 11.Set the return secret as o1
- 1201 12.Return TPM_SUCCESS

1202 **10.3 TPM_UnBind**1203 **Start of informative comment:**

1204 TPM_UnBind takes the data blob that is the result of a Tspi_Data_Bind command and
1205 decrypts it for export to the User. The caller must authorize the use of the key that will
1206 decrypt the incoming blob.

1207 TPM_UnBind operates on a block-by-block basis, and has no notion of any relation between
1208 one block and another.

1209 **End of informative comment.**1210 **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.

1211 **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_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.

1212 **Description**

1213 TPM_UnBind SHALL operate on a single block only.

1214 **Actions**

1215 The TPM SHALL perform the following:

- 1216 1. If the inDataSize is 0 the TPM returns TPM_BAD_PARAMETER
- 1217 2. Validate the AuthData to use the key pointed to by keyHandle
- 1218 3. If the keyUsage field of the key referenced by keyHandle does not have the value
1219 TPM_KEY_BIND or TPM_KEY_LEGACY, the TPM must return the error code
1220 TPM_INVALID_KEYUSAGE
- 1221 4. Decrypt the inData using the key pointed to by keyHandle
- 1222 5. if (keyHandle -> encScheme does not equal TPM_ES_RSAESOAEP_SHA1_MGF1) and
1223 (keyHandle -> keyUsage equals TPM_KEY_LEGACY),
 - 1224 a. The payload does not have TPM specific markers to validate, so no consistency check
1225 can be performed.
 - 1226 b. Set the output parameter outData to the value of the decrypted value of inData.
1227 (Padding associated with the encryption wrapping of inData SHALL NOT be returned.)
 - 1228 c. Set the output parameter outDataSize to the size of outData, as deduced from the
1229 decryption process.
- 1230 6. else
 - 1231 a. Interpret the decrypted data under the assumption that it is a TPM_BOUND_DATA
1232 structure, and validate that the payload type is TPM_PT_BIND

- 1233 b. Set the output parameter `outData` to the value of `TPM_BOUND_DATA` ->
1234 `payloadData`. (Other parameters of `TPM_BOUND_DATA` SHALL NOT be returned.
1235 Padding associated with the encryption wrapping of `inData` SHALL NOT be returned.)
- 1236 c. Set the output parameter `outDataSize` to the size of `outData`, as deduced from the
1237 decryption process and the interpretation of `TPM_BOUND_DATA`.
- 1238 7. Return the output parameters.

1239 **10.4 TPM_CreateWrapKey**

1240 **Start of informative comment:**

1241 The TPM_CreateWrapKey command both generates and creates a secure storage bundle for
1242 asymmetric keys.

1243 The newly created key can be locked to a specific PCR value by specifying a set of PCR
1244 registers.

1245 **End of informative comment.**

1246 **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 sealed data.
6	20	3S	20	TPM_ENCAUTH	dataMigrationAuth	Encrypted migration AuthData for the sealed data.
7	<>	4S	<>	TPM_KEY	keyInfo	Information about key to be created, pubkey.keyLength and keyInfo.encData elements are 0. MAY be TPM_KEY12
8	4			TPM_AUTHHANDLE	authHandle	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.

1247 **Outgoing Operands and Sizes**

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

1248 **Actions**

1249 The TPM SHALL do the following:

- 1250 1. Validate the AuthData to use the key pointed to by parentHandle. Return
1251 TPM_AUTHFAIL on any error.
- 1252 2. Validate the session type for parentHandle is OSAP.
- 1253 3. If the TPM is not designed to create a key of the type requested in keyInfo, return the
1254 error code TPM_BAD_KEY_PROPERTY
- 1255 4. Verify that parentHandle->keyUsage equals TPM_KEY_STORAGE
- 1256 5. If parentHandle -> keyFlags -> migratable is TRUE and keyInfo -> keyFlags -> migratable
1257 is FALSE then return TPM_INVALID_KEYUSAGE
- 1258 6. Validate key parameters
- 1259 a. keyInfo -> keyUsage MUST NOT be TPM_KEY_IDENTITY or
1260 TPM_KEY_AUTHCHANGE. If it is, return TPM_INVALID_KEYUSAGE
- 1261 b. If keyInfo -> keyFlags -> migrateAuthority is TRUE then return
1262 TPM_INVALID_KEYUSAGE
- 1263 7. If TPM_PERMANENT_FLAGS -> FIPS is TRUE then
- 1264 a. If keyInfo -> keySize is less than 1024 return TPM_NOTFIPS
- 1265 b. If keyInfo -> authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS
- 1266 c. If keyInfo -> keyUsage specifies TPM_KEY_LEGACY return TPM_NOTFIPS
- 1267 8. If keyInfo -> keyUsage equals TPM_KEY_STORAGE or TPM_KEY_MIGRATE
- 1268 i. algorithmID MUST be TPM_ALG_RSA
- 1269 ii. encScheme MUST be TPM_ES_RSAESOAEP_SHA1_MGF1
- 1270 iii. sigScheme MUST be TPM_SS_NONE
- 1271 iv. key size MUST be 2048
- 1272 9. Determine the version of key
- 1273 a. If keyInfo -> ver is 1.1
- 1274 i. Set V1 to 1
- 1275 ii. Map wrappedKey to a TPM_KEY structure
- 1276 iii. Validate all remaining TPM_KEY structures
- 1277 b. Else if keyInfo -> tag is TPM_TAG_KEY12
- 1278 i. Set V1 to 2
- 1279 ii. Map wrappedKey to a TPM_KEY12 structure
- 1280 iii. Validate all remaining TPM_KEY12 structures
- 1281 10. Create DU1 by decrypting dataUsageAuth according to the ADIP indicated by
1282 authHandle

- 1283 11. Create DM1 by decrypting dataMigrationAuth according to the ADIP indicated by
1284 authHandle
- 1285 12. Set continueAuthSession to FALSE
- 1286 13. Generate asymmetric key according to algorithm information in keyInfo
- 1287 14. Fill in the wrappedKey structure with information from the newly generated key.
- 1288 a. Set wrappedKey -> encData -> usageAuth to DU1
- 1289 b. If the KeyFlags -> migratable bit is set to 1, the wrappedKey -> encData ->
1290 migrationAuth SHALL contain the decrypted value from dataMigrationAuth.
- 1291 c. If the KeyFlags -> migratable bit is set to 0, the wrappedKey -> encData ->
1292 migrationAuth SHALL be set to the value tpmProof
- 1293 15. If keyInfo->PCRInfoSize is non-zero
- 1294 a. If V1 is 1
- 1295 i. Set wrappedKey -> pcrInfo to a TPM_PCR_INFO structure using the pcrSelection
1296 to indicate the PCR's in use
- 1297 b. Else
- 1298 i. Set wrappedKey -> pcrInfo to a TPM_PCR_INFO_LONG structure
- 1299 c. Set wrappedKey -> pcrInfo to keyInfo -> pcrInfo
- 1300 d. Set wrappedKey -> digestAtCreation to the TPM_COMPOSITE_HASH indicated by
1301 creationPCRSelection
- 1302 e. If V1 is 2 set wrappedKey -> localityAtCreation to TPM_STANY_DATA -> locality
- 1303 16. Encrypt the private portions of the wrappedKey structure using the key in parentHandle
- 1304 17. Return the newly generated key in the wrappedKey parameter

1305 **10.5 TPM_LoadKey2**1306 **Start of informative comment:**

1307 Before the TPM can use a key to either wrap, unwrap, unbind, seal, unseal, sign or perform
1308 any other action, it needs to be present in the TPM. The TPM_LoadKey2 function loads the
1309 key into the TPM for further use.

1310 The TPM assigns the key handle. The TPM always locates a loaded key by use of the handle.
1311 The assumption is that the handle may change due to key management operations. It is the
1312 responsibility of upper level software to maintain the mapping between handle and any
1313 label used by external software.

1314 To permit this mapping between handle and upper software labels (called key handle
1315 virtualization), the key handle returned by TPM_LoadKey2 must not be included in the
1316 response HMAC. This may cause problems if several keys are authorized using the same
1317 authorization data. Care should be taken to assign different authorization data to each key.

1318 This command has the responsibility of enforcing restrictions on the use of keys. For
1319 example, when attempting to load a STORAGE key it will be checked for the restrictions on
1320 a storage key (2048 size etc.).

1321 The load command must maintain a record of whether any previous key in the key
1322 hierarchy was bound to a PCR using parentPCRStatus.

1323 The flag parentPCRStatus enables the possibility of checking that a platform passed
1324 through some particular state or states before finishing in the current state. A grandparent
1325 key could be linked to state-1, a parent key could be linked to state-2, and a child key could be
1326 linked to state-3, for example. The use of the child key then indicates that the platform
1327 passed through states 1 and 2 and is currently in state 3, in this example. TPM_Startup
1328 with stType == TPM_ST_CLEAR indicates that the platform has been reset, so the platform
1329 has not passed through the previous states. Hence keys with parentPCRStatus==TRUE
1330 must be unloaded if TPM_Startup is issued with stType == TPM_ST_CLEAR.

1331 If a TPM_KEY structure has been decrypted AND the integrity test using "pubDataDigest"
1332 has passed AND the key is non-migratory, the key must have been created by the TPM. So
1333 there is every reason to believe that the key poses no security threat to the TPM. While there
1334 is no known attack from a rogue migratory key, there is a desire to verify that a loaded
1335 migratory key is a real key, arising from a general sense of unease about execution of
1336 arbitrary data as a key. Ideally a consistency check would consist of an encrypt/decrypt
1337 cycle, but this may be expensive. For RSA keys, it is therefore suggested that the
1338 consistency test consists of dividing the supposed RSA product by the supposed RSA prime,
1339 and checking that there is no remainder.

1340 **End of informative comment.**

1341 **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.

1342 **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.

1343 **Actions**

1344 The TPM SHALL perform the following steps:

- 1345 1. Validate the command and the parameters using parentAuth and parentHandle ->
1346 usageAuth
- 1347 2. If parentHandle -> keyUsage is NOT TPM_KEY_STORAGE return
1348 TPM_INVALID_KEYUSAGE
- 1349 3. If the TPM is not designed to operate on a key of the type specified by inKey, return the
1350 error code TPM_BAD_KEY_PROPERTY
- 1351 4. The TPM MUST handle both TPM_KEY and TPM_KEY12 structures
- 1352 5. Decrypt the inKey -> privkey to obtain TPM_STORE_ASYMKEY structure using the key
1353 in parentHandle

- 1354 6. Validate the integrity of inKey and decrypted TPM_STORE_ASYMKEY
- 1355 a. Reproduce inKey -> TPM_STORE_ASYMKEY -> pubDataDigest using the fields of
- 1356 inKey, and check that the reproduced value is the same as pubDataDigest
- 1357 7. Validate the consistency of the key and it's key usage.
- 1358 a. If inKey -> keyFlags -> migratable is TRUE, the TPM SHALL verify consistency of the
- 1359 public and private components of the asymmetric key pair. If inKey -> keyFlags ->
- 1360 migratable is FALSE, the TPM MAY verify consistency of the public and private
- 1361 components of the asymmetric key pair. The consistency of an RSA key pair MAY be
- 1362 verified by dividing the supposed (P*Q) product by a supposed prime and checking that
- 1363 there is no remainder.
- 1364 b. If inKey -> keyUsage is TPM_KEY_IDENTITY, verify that inKey->keyFlags->migratable
- 1365 is FALSE. If it is not, return TPM_INVALID_KEYUSAGE
- 1366 c. If inKey -> keyUsage is TPM_KEY_AUTHCHANGE, return TPM_INVALID_KEYUSAGE
- 1367 d. If inKey -> keyFlags -> migratable equals 0 then verify that TPM_STORE_ASYMKEY -
- 1368 > migrationAuth equals TPM_PERMANENT_DATA -> tpmProof
- 1369 e. Validate the mix of encryption and signature schemes
- 1370 f. If TPM_PERMANENT_FLAGS -> FIPS is TRUE then
- 1371 i. If keyInfo -> keySize is less than 1024 return TPM_NOTFIPS
- 1372 ii. If keyInfo -> authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS
- 1373 iii. If keyInfo -> keyUsage specifies TPM_KEY_LEGACY return TPM_NOTFIPS
- 1374 g. If inKey -> keyUsage is TPM_KEY_STORAGE or TPM_KEY_MIGRATE
- 1375 i. algorithmID MUST be TPM_ALG_RSA
- 1376 ii. Key size MUST be 2048
- 1377 iii. sigScheme MUST be TPM_SS_NONE
- 1378 h. If inKey -> keyUsage is TPM_KEY_IDENTITY
- 1379 i. algorithmID MUST be TPM_ALG_RSA
- 1380 ii. Key size MUST be 2048
- 1381 iii. encScheme MUST be TPM_ES_NONE
- 1382 i. If the decrypted inKey -> pcrInfo is NULL,
- 1383 i. The TPM MUST set the internal indicator to indicate that the key is not using any
- 1384 PCR registers.
- 1385 j. Else
- 1386 i. The TPM MUST store pcrInfo in a manner that allows the TPM to calculate a
- 1387 composite hash whenever the key will be in use
- 1388 ii. The TPM MUST handle both version 1.1 TPM_PCR_INFO and 1.2
- 1389 TPM_PCR_INFO_LONG structures according to the type of TPM_KEY structure
- 1390 (1) The TPM MUST validate the TPM_PCR_INFO or TPM_PCR_INFO_LONG
- 1391 structures

- 1392 8. Perform any processing necessary to make TPM_STORE_ASYMKEY key available for
1393 operations
- 1394 9. Load key and key information into internal memory of the TPM. If insufficient memory
1395 exists return error TPM_NOSPACE.
- 1396 10. Assign inKeyHandle according to internal TPM rules.
- 1397 11. Set InKeyHandle -> parentPCRStatus to parentHandle -> parentPCRStatus.
- 1398 12. If ParentHandle indicates that it is using PCR registers, then set inKeyHandle ->
1399 parentPCRStatus to TRUE.

10.6 TPM_GetPubKey

Start of informative comment:

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

End of informative comment.

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: 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.

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_GetPubKey.
4	<>	3S	<>	TPM_PUBKEY	pubKey	Public portion of key in keyHandle.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	Authorization. HMAC key: key.usageAuth.

Actions

The TPM SHALL perform the following steps:

1. If tag = TPM_TAG_RQU_AUTH1_COMMAND then
 - a. Validate the command parameters using keyHandle -> usageAuth, on error return TPM_AUTHFAIL
2. Else

- 1414 a. Verify that keyHandle -> authDataUsage is TPM_AUTH_PRIV_USE_ONLY or
1415 TPM_AUTH_NEVER, on error return TPM_AUTHFAIL
- 1416 3. If keyHandle == TPM_KH_SRK then
- 1417 a. If TPM_PERMANENT_FLAGS -> readSRKPub is FALSE then return
1418 TPM_INVALID_KEYHANDLE
- 1419 4. If keyHandle -> pcrInfoSize is not 0
- 1420 a. If keyHandle -> keyFlags has pcrIgnoredOnRead set to FALSE
- 1421 i. Create a digestAtRelease according to the specified PCR registers and compare to
1422 keyHandle -> digestAtRelease and if a mismatch return TPM_WRONGPCRVAL
- 1423 ii. If specified validate any locality requests
- 1424 5. Create a TPM_PUBKEY structure and return

1425 **10.7 TPM_Sealx**1426 **Start of informative comment:**

1427 The SEALX command works exactly like the SEAL command with the additional
1428 requirement of encryption for the inData parameter. This command also places in the
1429 sealed blob the information that the unseal also requires encryption.

1430 SEALX requires the use of 1.2 data structures. The actions are the same as SEAL without
1431 the checks for 1.1 data structure usage.

1432 **The method of incrementing the symmetric key counter value is different from that**
1433 **used by some standard crypto libraries (e.g. openssl, Java JCE) that increment the**
1434 **entire counter value. TPM users should be aware of this to avoid errors when the**
1435 **counter wraps.**End of informative comment.

1436 **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.

1437 **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.

1438 **Actions**

- 1439 1. Validate the authorization to use the key pointed to by keyHandle
- 1440 2. If the inDataSize is 0 the TPM returns TPM_BAD_PARAMETER
- 1441 3. If the keyUsage field of the key indicated by keyHandle does not have the value
- 1442 TPM_KEY_STORAGE, the TPM must return the error code TPM_INVALID_KEYUSAGE.
- 1443 4. If the keyHandle points to a migratable key then the TPM MUST return the error code
- 1444 TPM_INVALID_KEY_USAGE.
- 1445 5. Create S1 a TPM_STORED_DATA12 structure
- 1446 6. Set s1 -> encDataSize to 0
- 1447 7. Set s1 -> encData to all zeros
- 1448 8. Set s1 -> sealInfoSize to pcrInfoSize
- 1449 9. If pcrInfoSize is not 0 then
- 1450 a. Validate pcrInfo as a valid TPM_PCR_INFO_LONG structure, return TPM_BADINDEX
- 1451 on error
- 1452 b. Set s1 -> sealInfo -> creationPCRSelection to pcrInfo -> creationPCRSelection
- 1453 c. Set s1 -> sealInfo -> releasePCRSelection to pcrInfo -> releasePCRSelection
- 1454 d. Set s1 -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease
- 1455 e. Set s1 -> sealInfo -> localityAtRelease to pcrInfo -> localityAtRelease
- 1456 f. Create h2 the composite hash of the TPM_STCLEAR_DATA -> PCR selected by
- 1457 pcrInfo -> creationPCRSelection
- 1458 g. Set s1 -> sealInfo -> digestAtCreation to h2
- 1459 h. Set s1 -> sealInfo -> localityAtCreation to TPM_STANY_DATA -> localityModifier
- 1460 10. Create s2 a TPM_SEALED_DATA structure

- 1461 11. Create a1 by decrypting encAuth according to the ADIP indicated by authHandle.
- 1462 a. If authHandle indicates XOR encryption for the AuthData secrets
- 1463 i. Set s1 -> et to TPM_ET_XOR || TPM_ET_KEY
- 1464 (1) TPM_ET_KEY is added because TPM_Unseal uses zero as a special value
- 1465 indicating no encryption.
- 1466 b. Else
- 1467 i. Set S1 -> et to the algorithm indicated by authHandle
- 1468 12. The TPM provides NO validation of a1. Well-known values (like all zeros) are valid and
- 1469 possible.
- 1470 13. If authHandle indicates XOR encryption
- 1471 a. Use MGF1 to create string X2 of length inDataSize. The inputs to MGF1 are;
- 1472 authLastNonceEven, nonceOdd, "XOR", and authHandle -> sharedSecret. The four
- 1473 concatenated values form the Z value that is the seed for MFG1.
- 1474 b. Create o1 by XOR of inData and X2
- 1475 14. Else
- 1476 a. Create o1 by decrypting inData using the algorithm indicated by authHandle
- 1477 b. Key is from authHandle -> sharedSecret
- 1478 c. CTR is SHA-1 of (authLastNonceEven || nonceOdd)
- 1479 15. Create s2 a TPM_SEALED_DATA structure
- 1480 a. Set s2 -> payload to TPM_PT_SEAL
- 1481 b. Set s2 -> tpmProof to TPM_PERMANENT_DATA -> tpmProof
- 1482 c. Create h3 the SHA-1 of s1
- 1483 d. Set s2 -> storedDigest to h3
- 1484 e. Set s2 -> authData to a1
- 1485 f. Set s2 -> dataSize to inDataSize
- 1486 g. Set s2 -> data to o1
- 1487 16. Validate that the size of s2 can be encrypted by the key pointed to by keyHandle, return
- 1488 TPM_BAD_DATASIZE on error
- 1489 17. Create s3 the encryption of s2 using the key pointed to by keyHandle
- 1490 18. Set continueAuthSession to FALSE
- 1491 19. Set s1 -> encDataSize to the size of s3
- 1492 20. Set s1 -> encData to s3
- 1493 21. Return s1 as sealedData

1494 **11. Migration**

1495 **Start of informative comment:**

1496 The migration of a key from one TPM to another is a vital aspect to many use models of the
1497 TPM. The migration commands are the commands that allow this operation to occur.

1498 There are two types of migratable keys, the version 1.1 migratable keys and the version 1.2
1499 certifiable migratable keys.

1500 **End of informative comment.**

1501 **11.1 TPM_CreateMigrationBlob**

1502 **Start of informative comment:**

1503 The TPM_CreateMigrationBlob command implements the first step in the process of moving
1504 a migratable key to a new parent or platform. Execution of this command requires
1505 knowledge of the migrationAuth field of the key to be migrated.

1506 Migrate mode is generally used to migrate keys from one TPM to another for backup,
1507 upgrade or to clone a key on another platform. To do this, the TPM needs to create a data
1508 blob that another TPM can deal with. This is done by loading in a backup public key that
1509 will be used by the TPM to create a new data blob for a migratable key.

1510 The TPM Owner does the selection and authorization of migration public keys at any time
1511 prior to the execution of TPM_CreateMigrationBlob by performing the
1512 TPM_AuthorizeMigrationKey command.

1513 IReWrap mode is used directly to move the key to a new parent (on either this platform or
1514 another). The TPM simply re-encrypts the key using a new parent, and outputs a normal
1515 encrypted element that can be subsequently used by a TPM_LoadKey command.

1516 TPM_CreateMigrationBlob implicitly cannot be used to migrate a non-migratory key. No
1517 explicit check is required. Only the TPM knows tpmProof. Therefore, it is impossible for the
1518 caller to submit an AuthData value equal to tpmProof and migrate a non-migratory key.

1519 **End of informative comment.**

1520 **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.

1521

1522 **Outgoing Operands and Sizes**

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

1523 **Description**

1524 The TPM does not check the PCR values when migrating values locked to a PCR.

1525 The second authorization session (using entityAuth) MUST be OIAP because OSAP does not
1526 have a suitable entityType

1527 **Actions**

- 1528 1. Validate that parentAuth authorizes the use of the key pointed to by parentHandle.
- 1529 2. Validate that parentHandle -> keyUsage is TPM_KEY_STORAGE, if not return
1530 TPM_INVALID_KEYUSAGE
- 1531 3. Create d1 a TPM_STORE_ASYMKEY structure by decrypting encData using the key
1532 pointed to by parentHandle.
 - 1533 a. Verify that d1 -> payload is TPM_PT_ASYM.
- 1534 4. Validate that entityAuth authorizes the migration of d1. The validation MUST use d1 ->
1535 migrationAuth as the secret.
- 1536 5. Verify that the digest within migrationKeyAuth is legal for this TPM and public key
- 1537 6. If migrationType == TPM_MS_MIGRATE the TPM SHALL perform the following actions:
 - 1538 a. Build two byte arrays, K1 and K2:
 - 1539 i. K1 = d1.privKey[0..19] (d1.privKey.keyLength + 16 bytes of d1.privKey.key),
1540 sizeof(K1) = 20

- 1541 ii. K2 = d1.privKey[20..131] (position 16-127 of d1 . privKey.key), sizeof(K2) = 112
- 1542 b. Build M1 a TPM_MIGRATE_ASYMKEY structure
- 1543 i. TPM_MIGRATE_ASYMKEY.payload = TPM_PT_MIGRATE
- 1544 ii. TPM_MIGRATE_ASYMKEY.usageAuth = d1.usageAuth
- 1545 iii. TPM_MIGRATE_ASYMKEY.pubDataDigest = d1. pubDataDigest
- 1546 iv. TPM_MIGRATE_ASYMKEY.partPrivKeyLen = 112 – 127.
- 1547 v. TPM_MIGRATE_ASYMKEY.partPrivKey = K2
- 1548 c. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the
- 1549 OAEP encoding of m using OAEP parameters of
- 1550 i. m = M1 the TPM_MIGRATE_ASYMKEY structure
- 1551 ii. pHash = d1->migrationAuth
- 1552 iii. seed = s1 = K1
- 1553 d. Create r1 a random value from the TPM RNG. The size of r1 MUST be the size of o1.
- 1554 Return r1 in the Random parameter.
- 1555 e. Create x1 by XOR of o1 with r1
- 1556 f. Copy r1 into the output field “random”.
- 1557 g. Encrypt x1 with the migration public key included in migrationKeyAuth.
- 1558 7. If migrationType == TPM_MS_REWRAP the TPM SHALL perform the following actions:
- 1559 a. Rewrap the key using the public key in migrationKeyAuth, keeping the existing
- 1560 contents of that key.
- 1561 b. Set randomSize to 0 in the output parameter array
- 1562 8. Else
- 1563 a. Return TPM_BAD_PARAMETER

1564 **11.2 TPM_ConvertMigrationBlob**

1565 **Start of informative comment:**

1566 This command takes a migration blob and creates a normal wrapped blob. The migrated
1567 blob must be loaded into the TPM using the normal TPM_LoadKey function.

1568 Note that the command migrates private keys, only. The migration of the associated public
1569 keys is not specified by TPM because they are not security sensitive. Migration of the
1570 associated public keys may be specified in a platform specific specification. A TPM_KEY
1571 structure must be recreated before the migrated key can be used by the target TPM in a
1572 TPM_LoadKey command.

1573 **End of informative comment.**

1574 **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

1575

1576 **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

1577 **Action**

1578 The TPM SHALL perform the following:

- 1579 1. Validate the AuthData to use the key in parentHandle
- 1580 2. If the keyUsage field of the key referenced by parentHandle does not have the value
1581 TPM_KEY_STORAGE, the TPM must return the error code TPM_INVALID_KEYUSAGE
- 1582 3. Create d1 by decrypting the inData area using the key in parentHandle
- 1583 4. Create o1 by XOR d1 and random parameter
- 1584 5. Create m1 a TPM_MIGRATE_ASYMKEY structure, seed and pHash by OAEP decoding o1
- 1585 6. Create k1 by combining seed and the TPM_MIGRATE_ASYMKEY -> partPrivKey field
- 1586 7. Create d2 a TPM_STORE_ASYMKEY structure
 - 1587 a. Verify that m1 -> payload == TPM_PT_MIGRATE
 - 1588 b. Set d2 -> payload = TPM_PT_ASYM
 - 1589 c. Set d2 -> usageAuth to m1 -> usageAuth
 - 1590 d. Set d2 -> migrationAuth to pHash
 - 1591 e. Set d2 -> pubDataDigest to m1 -> pubDataDigest
 - 1592 f. Set d2 -> privKey field to k1
- 1593 8. Create outData using the key in parentHandle to perform the encryption

1594 **11.3 TPM_AuthorizeMigrationKey**

1595 **Start of informative comment:**

1596 This command creates an authorization blob, to allow the TPM owner to specify which
1597 migration facility they will use and allow users to migrate information without further
1598 involvement with the TPM owner.

1599 It is the responsibility of the TPM Owner to determine whether migrationKey is appropriate
1600 for migration. The TPM checks just the cryptographic strength of migrationKey.

1601 **End of informative comment.**

1602 **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.

1603 **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.

1604

1605 **Action**

1606 The TPM SHALL perform the following:

- 1607 1. Check that the cryptographic strength of migrationKey is at least that of a 2048 bit RSA
1608 key. If migrationKey is an RSA key, this means that migrationKey MUST be 2048 bits or
1609 greater
- 1610 2. Validate the AuthData to use the TPM by the TPM Owner
- 1611 3. Create a f1 a TPM_MIGRATIONKEYAUTH structure
- 1612 4. Verify that migrationKey-> algorithmParms -> encScheme is
1613 TPM_ES_RSAESOAEP_SHA1_MGF1, and return the error code
1614 TPM_INAPPROPRIATE_ENC if it is not
- 1615 5. Set f1 -> migrationKey to the input migrationKey
- 1616 6. Set f1 -> migrationScheme to the input migrationScheme
- 1617 7. Create v1 by concatenating (migrationKey || migrationScheme ||
1618 TPM_PERMANENT_DATA -> tpmProof)
- 1619 8. Create h1 by performing a SHA-1 hash of v1
- 1620 9. Set f1 -> digest to h1
- 1621 10. Return f1 as outData

1622 **11.4 TPM_MigrateKey**

1623 **Start of informative comment:**

1624 The TPM_MigrateKey command performs the function of a migration authority.

1625 The command is relatively simple; it just decrypts the input packet (coming from
1626 TPM_CreateMigrationBlob or TPM_CMK_CreateBlob) and then re-encrypts it with the input
1627 public key. The output of this command would then be sent to TPM_ConvertMigrationBlob
1628 or TPM_CMK_ConvertMigration on the target TPM.

1629 TPM_MigrateKey does not make ANY assumptions about the contents of the encrypted blob.
1630 Since it does not have the XOR string, it cannot actually determine much about the key
1631 that is being migrated.

1632 This command exists to permit the TPM to be a migration authority. If used in this way, it is
1633 expected that the physical security of the system containing the TPM and the AuthData
1634 value for the MA key would be tightly controlled.

1635 To prevent the execution of this command using any other key as a parent key, this
1636 command works only if keyUsage for maKeyHandle is TPM_KEY_MIGRATE.

1637 **End of informative comment.**

1638 **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.

1639 **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

1640 **Actions**

- 1641 1. Validate that keyAuth authorizes the use of the key pointed to by maKeyHandle
- 1642 2. The TPM validates that the key pointed to by maKeyHandle has a key usage value of
1643 TPM_KEY_MIGRATE, and that the allowed encryption scheme is
1644 TPM_ES_RSAESOAEP_SHA1_MGF1.
- 1645 3. The TPM validates that pubKey is of a size supported by the TPM and that its size is
1646 consistent with the input blob and maKeyHandle.
- 1647 4. The TPM decrypts inData and re-encrypts it using pubKey.

1648 **11.5 TPM_CMK_SetRestrictions**

1649 **Start of informative comment:**

1650 This command is used by the Owner to dictate the usage of a certified-migration key with
1651 delegated authorization (authorization other than actual owner authorization).

1652 This command is provided for privacy reasons and must not itself be delegated, because a
1653 certified-migration-key may involve a contractual relationship between the Owner and an
1654 external entity.

1655 Since restrictions are validated at DSAP session use, there is no need to invalidate DSAP
1656 sessions when the restriction value changes.

1657 **End of informative comment.**

1658 **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

1659 **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.

1660 **Description**

1661 TPM_PERMANENT_DATA -> restrictDelegate is used as follows

- 1662 1. If the session type is TPM_PID_DSAP and TPM_KEY -> keyFlags -> migrateAuthority is
1663 TRUE
- 1664 a. If
- 1665 TPM_KEY_USAGE is TPM_KEY_SIGNING and restrictDelegate ->
1666 TPM_CMK_DELEGATE_SIGNING is TRUE, or
- 1667 TPM_KEY_USAGE is TPM_KEY_STORAGE and restrictDelegate ->
1668 TPM_CMK_DELEGATE_STORAGE is TRUE, or
- 1669 TPM_KEY_USAGE is TPM_KEY_BIND and restrictDelegate -> TPM_CMK_DELEGATE_BIND
1670 is TRUE, or
- 1671 TPM_KEY_USAGE is TPM_KEY_LEGACY and restrictDelegate ->
1672 TPM_CMK_DELEGATE_LEGACY is TRUE, or
- 1673 TPM_KEY_USAGE is TPM_KEY_MIGRATE and restrictDelegate ->
1674 TPM_CMK_DELEGATE_MIGRATE is TRUE
- 1675 then the key can be used.
- 1676 b. Else return TPM_INVALID_KEYUSAGE.

1677 **Actions**

- 1678 1. Validate the ordinal and parameters using TPM Owner authentication, return
1679 TPM_AUTHFAIL on error
- 1680 2. Set TPM_PERMANENT_DATA -> TPM_CMK_DELEGATE -> restrictDelegate = restriction
- 1681 3. Return TPM_SUCCESS

1682 **11.6 TPM_CMK_ApproveMA**

1683 **Start of informative comment:**

1684 This command creates an authorization ticket, to allow the TPM owner to specify which
1685 Migration Authorities they approve and allow users to create certified-migration-keys
1686 without further involvement with the TPM owner.

1687 It is the responsibility of the TPM Owner to determine whether a particular Migration
1688 Authority is suitable to control migration

1689 **End of informative comment.**

1690 **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.

1691 **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.

1692 **Action**

1693 The TPM SHALL perform the following:

- 1694 1. Validate the AuthData to use the TPM by the TPM Owner
- 1695 2. Create M2 a TPM_CMK_MA_APPROVAL structure

- 1696 a. Set M2 ->migrationAuthorityDigest to migrationAuthorityDigest
- 1697 3. Set outData = HMAC(M2) using tpmProof as the secret
- 1698 4. Return TPM_SUCCESS

1699 **11.7 TPM_CMK_CreateKey**

1700 **Start of informative comment:**

1701 The TPM_CMK_CreateKey command both generates and creates a secure storage bundle for
1702 asymmetric keys whose migration is controlled by a migration authority.

1703 TPM_CMK_CreateKey is very similar to TPM_CreateWrapKey, but: (1) the resultant key must
1704 be a migratable key and can be migrated only by TPM_CMK_CreateBlob; (2) the command is
1705 Owner authorized via a ticket.

1706 TPM_CMK_CreateKey creates an otherwise normal migratable key except that (1)
1707 migrationAuth is an HMAC of the migration authority and the new key's public key, signed
1708 by tpmProof (instead of being tpmProof); (2) the migrationAuthority bit is set TRUE; (3) the
1709 payload type is TPM_PT_MIGRATE_RESTRICTED.

1710 The migration-selection/migration authority is specified by passing in a public key (actually
1711 the digests of one or more public keys, so more than one migration authority can be
1712 specified).

1713 **End of informative comment.**

1714 **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 sealed data.
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.

1715 **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.

1716 **Actions**

1717 The TPM SHALL do the following:

- 1718 1. Validate the AuthData to use the key pointed to by parentHandle. Return
1719 TPM_AUTHFAIL on any error
- 1720 2. Validate the session type for parentHandle is OSAP
- 1721 3. If the TPM is not designed to create a key of the type requested in keyInfo, return the
1722 error code TPM_BAD_KEY_PROPERTY
- 1723 4. Verify that parentHandle->keyUsage equals TPM_KEY_STORAGE
- 1724 5. Verify that parentHandle-> keyFlags-> migratable == FALSE and parentHandle->
1725 encData -> migrationAuth == tpmProof
- 1726 6. If keyInfo -> keyFlags -> migratable is FALSE, return TPM_INVALID_KEYUSAGE
- 1727 7. If keyInfo -> keyFlags -> migrateAuthority is FALSE , return TPM_INVALID_KEYUSAGE
- 1728 8. Verify that the migration authority is authorized
- 1729 a. Create M1 a TPM_CMK_MA_APPROVAL structure
- 1730 i. Set M1 ->migrationAuthorityDigest to migrationAuthorityDigest
- 1731 b. Verify that migrationAuthorityApproval == HMAC(M1) using tpmProof as the secret
1732 and return error TPM_MA_AUTHORITY on mismatch
- 1733 9. Validate key parameters
- 1734 a. keyInfo -> keyUsage MUST NOT be TPM_KEY_IDENTITY or
1735 TPM_KEY_AUTHCHANGE. If it is, return TPM_INVALID_KEYUSAGE
- 1736 10.If TPM_PERMANENT_FLAGS -> FIPS is TRUE then
- 1737 a. If keyInfo -> keySize is less than 1024 return TPM_NOTFIPS
- 1738 b. If keyInfo -> authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS

- 1739 c. If keyInfo -> keyUsage specifies TPM_KEY_LEGACY return TPM_NOTFIPS
- 1740 11.If keyInfo -> keyUsage equals TPM_KEY_STORAGE or TPM_KEY_MIGRATE
- 1741 a. algorithmID MUST be TPM_ALG_RSA
- 1742 b. encScheme MUST be TPM_ES_RSAESOAEP_SHA1_MGF1
- 1743 c. sigScheme MUST be TPM_SS_NONE
- 1744 d. key size MUST be 2048
- 1745 12.If keyInfo -> tag is NOT TPM_TAG_KEY12 return error TPM_INVALID_STRUCTURE
- 1746 13.Map wrappedKey to a TPM_KEY12 structure
- 1747 14.Create DU1 by decrypting dataUsageAuth according to the ADIP indicated by
- 1748 authHandle.
- 1749 15.Set continueAuthSession to FALSE
- 1750 16.Generate asymmetric key according to algorithm information in keyInfo
- 1751 17.Fill in the wrappedKey structure with information from the newly generated key.
- 1752 a. Set wrappedKey -> encData -> usageAuth to DU1
- 1753 b. Set wrappedKey -> encData -> payload to TPM_PT_MIGRATE_RESTRICTED
- 1754 c. Create thisPubKey, a TPM_PUBKEY structure containing wrappedKey's public key
- 1755 and algorithm parameters
- 1756 d. Create M2 a TPM_CMK_MIGAUTH structure
- 1757 i. Set M2 -> msaDigest to migrationAuthorityDigest
- 1758 ii. Set M2 -> pubKeyDigest to SHA-1 (thisPubKey)
- 1759 e. Set wrappedKey -> encData -> migrationAuth equal to HMAC(M2), using tpmProof as
- 1760 the shared secret
- 1761 18.If keyInfo->PCRInfoSize is non-zero
- 1762 a. Set wrappedKey -> pcrInfo to a TPM_PCR_INFO_LONG structure
- 1763 b. Set wrappedKey -> pcrInfo to keyInfo -> pcrInfo
- 1764 c. Set wrappedKey -> digestAtCreation to the TPM_COMPOSITE_HASH indicated by
- 1765 creationPCRSelection
- 1766 d. Set wrappedKey -> localityAtCreation to TPM_STANY_FLAGS -> localityModifier
- 1767 19.Encrypt the private portions of the wrappedKey structure using the key in parentHandle
- 1768 20.Return the newly generated key in the wrappedKey parameter

1769 **11.8 TPM_CMK_CreateTicket**1770 **Start of informative comment:**

1771 The TPM_CMK_CreateTicket command uses a public key to verify the signature over a
1772 digest.

1773 TPM_CMK_CreateTicket returns a ticket that can be used to prove to the same TPM that
1774 signature verification with a particular public key was successful.

1775 **End of informative comment.**1776 **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.

1777 **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.

1778 **Actions**

1779 The TPM SHALL do the following:

- 1780 1. Validate the TPM Owner authentication to use the command
- 1781 2. Validate that the key type and algorithm are correct
 - 1782 a. Validate that verificationKey -> algorithmParms -> algorithmID == TPM_ALG_RSA
 - 1783 b. Validate that verificationKey -> algorithmParms -> encScheme == TPM_ES_NONE
 - 1784 c. Validate that verificationKey -> algorithmParms -> sigScheme is
 - 1785 TPM_SS_RSASSAPKCS1v15_SHA1 or TPM_SS_RSASSAPKCS1v15_INFO
- 1786 3. Use verificationKey to verify that signatureValue is a valid signature on signedData, and
- 1787 return error TPM_BAD_SIGNATURE on mismatch
- 1788 4. Create M2 a TPM_CMK_SIGTICKET
 - 1789 a. Set M2 -> verKeyDigest to the SHA-1 (verificationKey)
 - 1790 b. Set M2 -> signedData to signedData
- 1791 5. Set sigTicket = HMAC(M2) signed by using tpmProof as the secret
- 1792 6. Return TPM_SUCCESS

1793 **11.9 TPM_CMK_CreateBlob**1794 **Start of informative comment:**

1795 TPM_CMK_CreateBlob command is very similar to TPM_CreateMigrationBlob, except that it:
1796 (1) uses an extra ticket (restrictedKeyAuth) instead of a migrationAuth authorization
1797 session; (2) uses the migration options TPM_MS_RESTRICT_MIGRATE or
1798 TPM_MS_RESTRICT_APPROVE; (3) produces a wrapped key blob whose migrationAuth is
1799 independent of tpmProof.

1800 If the destination (parent) public key is the MA, migration is implicitly permitted. Further
1801 checks are required if the MA is not the destination (parent) public key, and merely selects
1802 a migration destination: (1) sigTicket must prove that restrictTicket was signed by the MA;
1803 (2) restrictTicket must vouch that the target public key is approved for migration to the
1804 destination (parent) public key. (Obviously, this more complex method may also be used by
1805 an MA to approve migration to that MA.) In both cases, the MA must be one of the MAs
1806 implicitly listed in the migrationAuth of the target key-to-be-migrated.

1807 **End of informative comment.**

1808 Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_CreateBlob
4	4			TPM_KEY_HANDLE	parentHandle	Handle of the parent key that can decrypt encData.
5	2	2S	2	TPM_MIGRATE_SCHEME	migrationType	The migration type, either TPM_MS_RESTRICT_MIGRATE or TPM_MS_RESTRICT_APPROVE
6	<>	3S	<>	TPM_MIGRATIONKEYAUTH	migrationKeyAuth	Migration public key and its authorization 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, which is a TPM_CMK_AUTH structure if migration type is TPM_MS_RESTRICT_APPROVE
11	<>	8S	<>	BYTE[]	restrictTicket	Either a NULL parameter or 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, which is a TPM_HMAC structure if migration type is TPM_MS_RESTRICT_APPROVE.
13	<>	10S	<>	BYTE[]	sigTicket	Either a NULL parameter or 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.

1809 **Outgoing Operands and Sizes**

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

1810 **Description**

1811 The TPM does not check the PCR values when migrating values locked to a PCR.

1812 **Actions**

- 1813 1. Validate that parentAuth authorizes the use of the key pointed to by parentHandle.
- 1814 2. The TPM MAY verify that migrationType == migrationKeyAuth -> migrationScheme and
1815 return TPM_BAD_MODE on error.
- 1816 a. The TPM MAY ignore migrationType.
- 1817 3. Verify that parentHandle-> keyFlags-> migratable == FALSE and parentHandle->
1818 encData -> migrationAuth == tpmProof
- 1819 4. Create d1 by decrypting encData using the key pointed to by parentHandle.
- 1820 5. Verify that the digest within migrationKeyAuth is legal for this TPM and public key
- 1821 6. Verify that d1 -> payload == TPM_PT_MIGRATE_RESTRICTED or
1822 TPM_PT_MIGRATE_EXTERNAL
- 1823 7. Verify that the migration authorities in msaList are authorized to migrate this key
- 1824 a. Create M2 a TPM_CMK_MIGAUTH structure
- 1825 i. Set M2 -> msaDigest to SHA-1[msaList]
- 1826 ii. Set M2 -> pubKeyDigest to pubSourceKeyDigest
- 1827 b. Verify that d1 -> migrationAuth == HMAC(M2) using tpmProof as the secret and
1828 return error TPM_MA_AUTHORITY on mismatch
- 1829 8. If migrationKeyAuth -> migrationScheme == TPM_MS_RESTRICT_MIGRATE
- 1830 a. Verify that intended migration destination is an MA:

- 1831 i. For one of $n=1$ to $n=(\text{msaList} \rightarrow \text{MSAlist})$, verify that $\text{SHA-1}[\text{migrationKeyAuth} \rightarrow$
1832 $\text{migrationKey}] == \text{msaList} \rightarrow \text{migAuthDigest}[n]$
- 1833 b. Validate that the MA key is the correct type
- 1834 i. Validate that $\text{migrationKeyAuth} \rightarrow \text{migrationKey} \rightarrow \text{algorithmParms} \rightarrow$
1835 $\text{algorithmID} == \text{TPM_ALG_RSA}$
- 1836 ii. Validate that $\text{migrationKeyAuth} \rightarrow \text{migrationKey} \rightarrow \text{algorithmParms} \rightarrow \text{encScheme}$
1837 is an encryption scheme supported by the TPM
- 1838 iii. Validate that $\text{migrationKeyAuth} \rightarrow \text{migrationKey} \rightarrow \text{algorithmParms} \rightarrow \text{sigScheme}$
1839 is TPM_SS_NONE
- 1840 9. else If $\text{migrationKeyAuth} \rightarrow \text{migrationScheme} == \text{TPM_MS_RESTRICT_APPROVE}$
- 1841 a. Verify that the intended migration destination has been approved by the MSA:
- 1842 i. Verify that for one of the $n=1$ to $n=(\text{msaList} \rightarrow \text{MSAlist})$ values of $\text{msaList} \rightarrow$
1843 $\text{migAuthDigest}[n]$, $\text{sigTicket} == \text{HMAC}(V1)$ using tpmProof as the secret where $V1$
1844 is a TPM_CMK_SIGTICKET structure such that:
- 1845 (1) $V1 \rightarrow \text{verKeyDigest} = \text{msaList} \rightarrow \text{migAuthDigest}[n]$
- 1846 (2) $V1 \rightarrow \text{signedData} = \text{SHA-1}[\text{restrictTicket}]$
- 1847 ii. If $[\text{restrictTicket} \rightarrow \text{destinationKeyDigest}] \neq \text{SHA-1}[\text{migrationKeyAuth} \rightarrow$
1848 $\text{migrationKey}]$, return error $\text{TPM_MA_DESTINATION}$
- 1849 iii. If $[\text{restrictTicket} \rightarrow \text{sourceKeyDigest}] \neq \text{pubSourceKeyDigest}$, return error
1850 TPM_MA_SOURCE
- 1851 10. Else return with error TPM_BAD_PARAMETER .
- 1852 11. Build two bytes array, $K1$ and $K2$, using $d1$:
- 1853 a. $K1 = \text{TPM_STORE_ASYMKEY.privKey}[0..19]$
1854 $(\text{TPM_STORE_ASYMKEY.privKey.keyLength} + 16 \text{ bytes of}$
1855 $\text{TPM_STORE_ASYMKEY.privKey.key})$, $\text{sizeof}(K1) = 20$
- 1856 b. $K2 = \text{TPM_STORE_ASYMKEY.privKey}[20..131]$ (position 16-127 of
1857 $\text{TPM_STORE_ASYMKEY} . \text{privKey.key}$), $\text{sizeof}(K2) = 112$
- 1858 12. Build $M1$ a $\text{TPM_MIGRATE_ASYMKEY}$ structure
- 1859 a. $\text{TPM_MIGRATE_ASYMKEY.payload} = \text{TPM_PT_CMK_MIGRATE}$
- 1860 b. $\text{TPM_MIGRATE_ASYMKEY.usageAuth} = \text{TPM_STORE_ASYMKEY.usageAuth}$
- 1861 c. $\text{TPM_MIGRATE_ASYMKEY.pubDataDigest} = \text{TPM_STORE_ASYMKEY.pubDataDigest}$
- 1862 d. $\text{TPM_MIGRATE_ASYMKEY.partPrivKeyLen} = 112 - 127$.
- 1863 e. $\text{TPM_MIGRATE_ASYMKEY.partPrivKey} = K2$
- 1864 13. Create $o1$ (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP
1865 encoding of m using OAEP parameters m , $pHash$, and $seed$
- 1866 a. m is the previously created $M1$
- 1867 b. $pHash = \text{SHA-1}(\text{SHA-1}[\text{msaList}] || \text{pubSourceKeyDigest})$
- 1868 c. $seed = s1 =$ the previously created $K1$

- 1869 14. Create r1 a random value from the TPM RNG. The size of r1 MUST be the size of o1.
1870 Return r1 in the random parameter
- 1871 15. Create x1 by XOR of o1 with r1
- 1872 16. Copy r1 into the output field “random”
- 1873 17. Encrypt x1 with the migrationKeyAuth-> migrationKey

1874 **11.10TPM_CMK_ConvertMigration**

1875 **Start of informative comment:**

1876 TPM_CMK_ConvertMigration completes the migration of certified migration blobs.

1877 This command takes a certified migration blob and creates a normal wrapped blob with
1878 payload type TPM_PT_MIGRATE_EXTERNAL. The migrated blob must be loaded into the
1879 TPM using the normal TPM_LoadKey function.

1880 Note that the command migrates private keys, only. The migration of the associated public
1881 keys is not specified by TPM because they are not security sensitive. Migration of the
1882 associated public keys may be specified in a platform specific specification. A TPM_KEY
1883 structure must be recreated before the migrated key can be used by the target TPM in a
1884 TPM_LoadKey command.

1885 TPM_CMK_ConvertMigration checks that one of the MAs implicitly listed in the
1886 migrationAuth of the target key has approved migration of the target key to the destination
1887 (parent) key, and that the settings (flags etc.) in the target key are those of a CMK.

1888 **End of informative comment.**

1889 **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

1890 **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

1891 **Action**

- 1892 1. Validate the AuthData to use the key in parentHandle
- 1893 2. If the keyUsage field of the key referenced by parentHandle does not have the value
1894 TPM_KEY_STORAGE, the TPM must return the error code TPM_INVALID_KEYUSAGE
- 1895 3. Create d1 by decrypting the migratedKey -> encData area using the key in parentHandle
- 1896 4. Create o1 by XOR d1 and random parameter
- 1897 5. Create m1 a TPM_MIGRATE_ASYMKEY, seed and pHash by OAEP decoding o1
- 1898 6. Create migratedPubKey a TPM_PUBKEY structure corresponding to migratedKey
- 1899 a. Verify that pHash == SHA-1(SHA-1[msaList] || SHA-1(migratedPubKey)
- 1900 7. Create k1 by combining seed and the TPM_MIGRATE_ASYMKEY -> partPrivKey field
- 1901 8. Create d2 a TPM_STORE_ASYMKEY structure.
- 1902 a. Set the TPM_STORE_ASYMKEY -> privKey field to k1
- 1903 b. Set d2 -> usageAuth to m1 -> usageAuth
- 1904 c. Set d2 -> pubDataDigest to m1 -> pubDataDigest
- 1905 9. Verify that parentHandle-> keyFlags -> migratable == FALSE and parentHandle->
1906 encData -> migrationAuth == tpmProof
- 1907 10. Verify that m1 -> payload == TPM_PT_CMK_MIGRATE then set d2-> payload =
1908 TPM_PT_MIGRATE_EXTERNAL
- 1909 11. Verify that for one of the n=1 to n=(msaList -> MSAList) values of msaList ->
1910 migAuthDigest[n] sigTicket == HMAC (V1) using tpmProof as the secret where V1 is a
1911 TPM_CMK_SIGTICKET structure such that:
- 1912 a. V1 -> verKeyDigest = msaList -> migAuthDigest[n]
- 1913 b. V1 -> signedData = SHA-1[restrictTicket]

- 1914 12.Create parentPubKey, a TPM_PUBKEY structure corresponding to parentHandle
- 1915 13.If [restrictTicket -> destinationKeyDigest] != SHA-1(parentPubKey), return error
1916 TPM_MA_DESTINATION
- 1917 14.Verify that migratedKey is corresponding to d2
- 1918 15.If migratedKey -> keyFlags -> migratable is FALSE, and return error
1919 TPM_INVALID_KEYUSAGE
- 1920 16.If migratedKey -> keyFlags -> migrateAuthority is FALSE, return error
1921 TPM_INVALID_KEYUSAGE
- 1922 17.If [restrictTicket -> sourceKeyDigest] != SHA-1(migratedPubKey), return error
1923 TPM_MA_SOURCE
- 1924 18.Create M2 a TPM_CMK_MIGAUTH structure
- 1925 a. Set M2 -> msaDigest to SHA-1[msaList]
- 1926 b. Set M2 -> pubKeyDigest to SHA-1[migratedPubKey]
- 1927 19.Set d2 -> migrationAuth = HMAC(M2) using tpmProof as the secret
- 1928 20.Create outData using the key in parentHandle to perform the encryption

1929 **12. Maintenance Functions (optional)**

1930 **Start of informative comment:**

1931 When a maintenance archive is created with generateRandom FALSE, the maintenance blob
1932 is XOR encrypted with the owner authorization before encryption with the maintenance
1933 public key. This prevents the manufacturer from obtaining plaintext data. The receiving
1934 TPM must have the same owner authorization as the sending TPM in order to XOR decrypt
1935 the archive.

1936 When generateRandom is TRUE, the maintenance blob is XOR encrypted with random data,
1937 which is also returned. This permits someone trusted by the Owner to load the
1938 maintenance archive into the replacement platform in the absence of the Owner and
1939 manufacturer, without the Owner having to reveal information about his auth value. The
1940 receiving and sending TPM's may have different owner authorizations. The random data is
1941 transferred from the sending TPM owner to the receiving TPM owner out of band, so the
1942 maintenance blob remains hidden from the manufacturer.

1943 This is a typical maintenance sequence:

1944 1. Manufacturer:

- 1945 • generates maintenance key pair
- 1946 • gives public key to TPM1 owner

1947 2. TPM1: TPM_LoadManuMaintPub

- 1948 • load maintenance public key

1949 3. TPM1: TPM_CreateMaintenanceArchive

- 1950 • XOR encrypt with owner auth or random
- 1951 • encrypt with maintenance public key

1952 4. Manufacturer:

- 1953 • decrypt with maintenance private key
- 1954 • (still XOR encrypted with owner auth or random)
- 1955 • encrypt with TPM2 SRK public key

1956 5. TPM2: TPM_LoadMaintenanceArchive

- 1957 • decrypt with SRK private key
- 1958 • XOR decrypt with owner auth or random

1959 **End of informative comment.**

1960

1961

1962 **12.1 TPM_CreateMaintenanceArchive**

1963 **Start of informative comment:**

1964 This command creates the maintenance archive. It can only be executed by the owner, and
1965 may be shut off with the TPM_KillMaintenanceFeature command.

1966 **End of informative comment.**

1967 **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.

1968 **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.

1969 **Actions**

1970 Upon authorization being confirmed this command does the following:

- 1971 1. Validates that the TPM_PERMANENT_FLAGS -> allowMaintenance is TRUE. If it is
1972 FALSE, the TPM SHALL return TPM_DISABLED_CMD and exit this capability.
- 1973 2. Validates the TPM Owner AuthData.
- 1974 3. If the value of TPM_PERMANENT_DATA -> manuMaintPub is zero, the TPM MUST
1975 return the error code TPM_KEYNOTFOUND
- 1976 4. Build a1 a TPM_KEY structure using the SRK. The encData field is not a normal
1977 TPM_STORE_ASYMKEY structure but rather a TPM_MIGRATE_ASYMKEY structure built
1978 using the following actions.
- 1979 5. Build a TPM_STORE_PRIVKEY structure from the SRK. This privKey element should be
1980 132 bytes long for a 2K RSA key.
- 1981 6. Create k1 and k2 by splitting the privKey element created in step 4 into 2 parts. k1 is
1982 the first 20 bytes of privKey, k2 contains the remainder of privKey.
- 1983 7. Build m1 by creating and filling in a TPM_MIGRATE_ASYMKEY structure
- 1984 a. m1 -> usageAuth is set to TPM_PERMANENT_DATA -> tpmProof
- 1985 b. m1 -> pubDataDigest is set to the digest value of the SRK fields from step 4
- 1986 c. m1 -> payload is set to TPM_PT_MAINT
- 1987 d. m1 -> partPrivKey is set to k2
- 1988 8. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP
1989 encoding of m using OAEP parameters of
- 1990 a. m = TPM_MIGRATE_ASYMKEY structure (step 7)
- 1991 b. pHash = TPM_PERMANENT_DATA -> ownerAuth
- 1992 c. seed = s1 = k1 (step 6)
- 1993 9. If generateRandom = TRUE
- 1994 a. Create r1 by obtaining values from the TPM RNG. The size of r1 MUST be the same
1995 size as o1. Set random parameter to r1
- 1996 10.If generateRandom = FALSE
- 1997 a. Create r1 by applying MGF1 to the TPM Owner AuthData. The size of r1 MUST be the
1998 same size as o1. Set randomSize to 0.
- 1999 11.Create x1 by XOR of o1 with r1
- 2000 12.Encrypt x1 with the manuMaintPub key using the TPM_ES_RSAESOAEP_SHA1_MGF1
2001 encryption scheme.
- 2002 13.Set a1 -> encData to the encryption of x1
- 2003 14.Set TPM_PERMANENT_FLAGS -> maintenanceDone to TRUE
- 2004 15.Return a1 in the archive parameter

2005 **12.2 TPM_LoadMaintenanceArchive**

2006 **Start of informative comment:**

2007 This command loads in a Maintenance archive that has been massaged by the
2008 manufacturer to load into another TPM.

2009 If the maintenance archive was created using the owner authorization for XOR encryption,
2010 the current owner authorization must be used for decryption. The owner authorization does
2011 not change.

2012 If the maintenance archive was created using random data for the XOR encryption, the
2013 vendor specific arguments must include the random data. The owner authorization may
2014 change.

2015 **End of informative comment.**

2016 **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	Size 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.

2017

2018 **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

2019 **Descriptions**

2020 The maintenance mechanisms in the TPM MUST not require the TPM to hold a global
2021 secret. The definition of global secret is a secret value shared by more than one TPM.

2022 The TPME is not allowed to pre-store or use unique identifiers in the TPM for the purpose of
2023 maintenance. The TPM MUST NOT use the endorsement key for identification or encryption
2024 in the maintenance process. The maintenance process MAY use a TPM Identity to deliver
2025 maintenance information to specific TPM's.

2026 The maintenance process can only change the SRK, tpmProof and TPM Owner AuthData
2027 fields.

2028 The maintenance process can only access data in shielded locations where this data is
2029 necessary to validate the TPM Owner, validate the TPME and manipulate the blob

2030 The TPM MUST be conformant to the TPM specification, protection profiles and security
2031 targets after maintenance. The maintenance MAY NOT decrease the security values from
2032 the original security target.

2033 The security target used to evaluate this TPM MUST include this command in the TOE.

2034 **Actions**

2035 The TPM SHALL perform the following when executing the command

- 2036 1. Validate the TPM Owner's AuthData
- 2037 2. Validate that the maintenance information was sent by the TPME. The validation
2038 mechanism MUST use a strength of function that is at least the same strength of
2039 function as a digital signature performed using a 2048 bit RSA key.
- 2040 3. The packet MUST contain m2 as defined in section 12.1.
- 2041 4. Ensure that only the target TPM can interpret the maintenance packet. The protection
2042 mechanism MUST use a strength of function that is at least the same strength of
2043 function as a digital signature performed using a 2048 bit RSA key.
- 2044 5. Execute the actions of TPM_OwnerClear.

- 2045 6. Process the maintenance information
- 2046 a. Update the SRK
- 2047 i. Set the SRK usageAuth to be the same as the source TPM owner's AuthData
- 2048 b. Update TPM_PERMANENT_DATA -> tpmProof
- 2049 c. Update TPM_PERMANENT_DATA -> ownerAuth
- 2050 7. Set TPM_PERMANENT_FLAGS -> maintenanceDone to TRUE
- 2051

2052 **12.3 TPM_KillMaintenanceFeature**2053 **Informative Comments:**

2054 The TPM_KillMaintenanceFeature is a permanent action that prevents ANYONE from
2055 creating a maintenance archive. This action, once taken, is permanent until a new TPM
2056 Owner is set.

2057 This action is to allow those customers who do not want the maintenance feature to not
2058 allow the use of the maintenance feature.

2059 At the discretion of the Owner, it should be possible to kill the maintenance feature in such
2060 a way that the only way to recover maintainability of the platform would be to wipe out the
2061 root keys. This feature is mandatory in any TPM that implements the maintenance feature.

2062 **End informative Comment**2063 **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.

2064 **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.

2065 **Actions**

- 2066 1. Validate the TPM Owner AuthData
- 2067 2. Set the TPM_PERMANENT_FLAGS.allowMaintenance flag to FALSE.

2068 **12.4 TPM_LoadManuMaintPub**

2069 **Informative Comments:**

2070 The TPM_LoadManuMaintPub command loads the manufacturer’s public key for use in the
2071 maintenance process. The command installs manuMaintPub in PERMANENT data storage
2072 inside a TPM. Maintenance enables duplication of non-migratory data in protected storage.
2073 There is therefore a security hole if a platform is shipped before the maintenance public key
2074 has been installed in a TPM.

2075 The command is expected to be used before installation of a TPM Owner or any key in TPM
2076 protected storage. It therefore does not use authorization.

2077 **End of Informative Comments**

2078 **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

2079 **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

2080 **Description**

2081 The pubKey MUST specify an algorithm whose strength is not less than the RSA algorithm
2082 with 2048bit keys.

2083 pubKey SHOULD unambiguously identify the entity that will perform the maintenance
2084 process with the TPM Owner.

2085 TPM_PERMANENT_DATA -> manuMaintPub SHALL exist in a TPM-shielded location, only.

2086 If an entity (Platform Entity) does not support the maintenance process but issues a
2087 platform credential for a platform containing a TPM that supports the maintenance process,
2088 the value of TPM_PERMANENT_DATA -> manuMaintPub MUST be set to zero before the
2089 platform leaves the entity’s control. That is, this ordinal can only be run once, and used to
2090 either load the key or load a NULL key.

2091 **Actions**

2092 The first valid TPM_LoadManuMaintPub command received by a TPM SHALL

2093 1. Store the parameter pubKey as TPM_PERMANENT_DATA -> manuMaintPub.

2094 2. Set checksum to SHA-1 of (pubKey || antiReplay)

2095 3. Export the checksum

2096 4. Subsequent calls to TPM_LoadManuMaintPub SHALL return code
2097 TPM_DISABLED_CMD.

2098 **12.5 TPM_ReadManuMaintPub**

2099 **Informative Comments:**

2100 The TPM_ReadManuMaintPub command is used to check whether the manufacturer’s
2101 public maintenance key in a TPM has the expected value. This may be useful during the
2102 manufacture process. The command returns a digest of the installed key, rather than the
2103 key itself. This hinders discovery of the maintenance key, which may (or may not) be useful
2104 for manufacturer privacy.

2105 The command is expected to be used before installation of a TPM Owner or any key in TPM
2106 protected storage. It therefore does not use authorization.

2107 **End of Informative Comments**

2108 **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

2109 **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

2110 **Description**

2111 This command returns the hash of the antiReplay nonce and the previously loaded
2112 manufacturer’s maintenance public key.

2113 **Actions**

2114 The TPM_ReadManuMaintPub command SHALL

- 2115 1. Create “checksum” by concatenating data to form (TPM_PERMANENT_DATA ->
2116 manuMaintPub || antiReplay) and passing the concatenated data through SHA-1.
- 2117 2. Export the checksum

2118 **13. Cryptographic Functions**2119 **13.1 TPM_SHA1Start**2120 **Start of informative comment:**

2121 This capability starts the process of calculating a SHA-1 digest.

2122 The exposure of the SHA-1 processing is a convenience to platforms in a mode that do not
2123 have sufficient memory to perform SHA-1 themselves. As such, the use of SHA-1 is
2124 restrictive on the TPM.2125 The TPM may not allow any other types of processing during the execution of a SHA-1
2126 session. There is only one SHA-1 session active on a TPM. The exclusivity of a SHA-1
2127 context is due to the relatively large volatile buffer it requires in order to hold the
2128 intermediate results between the SHA-1 context commands. This buffer can be in
2129 contradiction to other command needs.2130 After the execution of TPM_SHA1Start, and prior to TPM_SHA1Complete or
2131 TPM_SHA1CompleteExtend, the receipt of any command other than TPM_SHA1Update will
2132 cause the invalidation of the SHA-1 session.2133 **End of informative comment.**2134 **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

2135 **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.

2136 **Description**2137 1. This capability prepares the TPM for a subsequent TPM_SHA1Update,
2138 TPM_SHA1Complete or TPM_SHA1CompleteExtend command. The capability SHALL
2139 open a thread that calculates a SHA-1 digest.

- 2140 2. After receipt of TPM_SHA1Start, and prior to the receipt of TPM_SHA1Complete or
2141 TPM_SHA1CompleteExtend, receipt of any command other than TPM_SHA1Update
2142 invalidates the SHA-1 session.
- 2143 a. If the command received is TPM_ExecuteTransport, the SHA-1 session invalidation is
2144 based on the wrapped command, not the TPM_ExecuteTransport ordinal.
- 2145 b. A SHA-1 thread (start, update, complete) MUST take place either completely outside
2146 a transport session or completely within a single transport session.

2147 **13.2 TPM_SHA1Update**2148 **Start of informative comment:**

2149 This capability inputs complete blocks of data into a pending SHA-1 digest. At the end of
2150 the process, the digest remains pending.

2151 **End of informative comment.**2152 **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

2153 **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

2154 **Description**

2155 This command SHALL incorporate complete blocks of data into the digest of an existing
2156 SHA-1 thread. Only integral numbers of complete blocks (64 bytes each) can be processed.

2157 **13.3 TPM_SHA1Complete**

2158 **Start of informative comment:**

2159 This capability terminates a pending SHA-1 calculation.

2160 **End of informative comment.**

2161 **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

2162 **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.

2163 **Description**

2164 This command SHALL incorporate a partial or complete block of data into the digest of an
 2165 existing SHA-1 thread, and terminate that thread. hashDataSize MAY have values in the
 2166 range of 0 through 64, inclusive.

2167 If the SHA-1 thread has received no bytes the TPM SHALL calculate the SHA-1 of the empty
 2168 buffer.

2169 **13.4 TPM_SHA1CompleteExtend**2170 **Start of informative comment:**

2171 This capability terminates a pending SHA-1 calculation and EXTENDS the result into a
2172 Platform Configuration Register using a SHA-1 hash process.

2173 This command is designed to complete a hash sequence and extend a PCR in memory-less
2174 environments.

2175 **End of informative comment.**2176 **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

2177 **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.

2178 **Description**

2179 This command SHALL incorporate a partial or complete block of data into the digest of an
2180 existing SHA-1 thread, EXTEND the resultant digest into a PCR, and terminate the SHA-1
2181 session. hashDataSize MAY have values in the range of 0 through 64, inclusive.

2182 The SHA-1 session MUST terminate even if the command returns an error, e.g.
2183 TPM_BAD_LOCALITY.

2184 **Actions**

- 2185 1. Map V1 to TPM_STANY_DATA
- 2186 2. Map L1 to V1 -> localityModifier
- 2187 3. If the current locality, held in L1, is not selected in TPM_PERMANENT_DATA -> pcrAttrib
2188 [pcrNum]. pcrExtendLocal, return TPM_BAD_LOCALITY

- 2189 4. Create H1 the TPM_DIGEST of the SHA-1 session ensuring that hashData, if any, is
2190 added to the SHA-1 session
- 2191 5. Perform the actions of TPM_Extend using H1 as the data and pcrNum as the PCR to
2192 extend

2193 **13.5 TPM_Sign**2194 **Start of informative comment:**

2195 The Sign command signs data and returns the resulting digital signature

2196 **End of informative comment.**2197 **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

2198 **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

2199 **Description**

2200 The TPM MUST support all values of areaToSignSize that are legal for the defined signature
 2201 scheme and key size. The maximum value of areaToSignSize is determined by the defined
 2202 signature scheme and key size.

2203 In the case of PKCS1v15_SHA1 the areaToSignSize MUST be TPM_DIGEST (the hash size of
2204 a SHA-1 operation - see 8.5.1 TPM_SS_RSASSAPKCS1v15_SHA1). In the case of
2205 PKCS1v15_DER the maximum size of areaToSign is k-11 octets, where k is limited by the
2206 key size (see TPM_SS_RSASSAPKCS1v15_DER).

2207 **Actions**

- 2208 1. The TPM validates the AuthData to use the key pointed to by keyHandle.
- 2209 2. If the areaToSignSize is 0 the TPM returns TPM_BAD_PARAMETER.
- 2210 3. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING or TPM_KEY_LEGACY, if not
2211 return the error code TPM_INVALID_KEYUSAGE
- 2212 4. The TPM verifies that the signature scheme and key size can properly sign the
2213 areaToSign parameter.
- 2214 5. If signature scheme is TPM_SS_RSASSAPKCS1v15_SHA1 then
 - 2215 a. Validate that areaToSignSize is 20 return TPM_BAD_PARAMETER on error
 - 2216 b. Set S1 to areaToSign
- 2217 6. Else if signature scheme is TPM_SS_RSASSAPKCS1v15_DER then
 - 2218 a. Validate that areaToSignSize is at least 11 bytes less than the key size, return
2219 TPM_BAD_PARAMETER on error
 - 2220 b. Set S1 to areaToSign
- 2221 7. else if signature scheme is TPM_SS_RSASSAPKCS1v15_INFO then
 - 2222 a. Create S2 a TPM_SIGN_INFO structure
 - 2223 b. Set S2 -> fixed to "SIGN"
 - 2224 c. Set S2 -> replay to nonceOdd
 - 2225 i. If nonceOdd is not present due to an unauthorized command return
2226 TPM_BAD_PARAMETER
 - 2227 d. Set S2 -> dataLen to areaToSignSize
 - 2228 e. Set S2 -> data to areaToSign
 - 2229 f. Set S1 to the SHA-1(S2)
- 2230 8. Else return TPM_INVALID_KEYUSAGE
- 2231 9. The TPM computes the signature, sig, using the key referenced by keyHandle using S1
2232 as the value to sign
- 2233 10. Return the computed signature in Sig

2234 **13.6 TPM_GetRandom**2235 **Start of informative comment:**

2236 TPM_GetRandom returns the next bytesRequested bytes from the random number
2237 generator to the caller.

2238 It is recommended that a TPM implement the RNG in a manner that would allow it to return
2239 RNG bytes such that the frequency of bytesRequested being more than the number of bytes
2240 available is an infrequent occurrence.

2241 **End of informative comment.**2242 **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

2243 **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

2244 **Actions**

- 2245 1. The TPM determines if amount bytesRequested is available from the TPM.
- 2246 2. Set randomBytesSize to the number of bytes available from the RNG. This number MAY
2247 be less than bytesRequested.
- 2248 3. Set randomBytes to the next randomBytesSize bytes from the RNG

2249 **13.7 TPM_StirRandom**

2250 **Start of informative comment:**

2251 TPM_StirRandom adds entropy to the RNG state.

2252 **End of informative comment.**

2253 **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 (<256)
5	<>	3S	<>	BYTE[]	inData	Data to add entropy to RNG state

2254 **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

2255 **Actions**

2256 The TPM updates the state of the current RNG using the appropriate mixing function.

2257 **13.8 TPM_CertifyKey**2258 **Start of informative comment:**

2259 The TPM_CertifyKey operation allows one key to certify the public portion of another key.

2260 A TPM identity key may be used to certify non-migratable keys but is not permitted to
2261 certify migratory keys or certified migration keys. As such, it allows the TPM to make the
2262 statement “this key is held in a TPM-shielded location, and it will never be revealed.” For
2263 this statement to have veracity, the Challenger must trust the policies used by the entity
2264 that issued the identity and the maintenance policy of the TPM manufacturer.

2265 Signing and legacy keys may be used to certify both migratable and non-migratable keys.
2266 Then the usefulness of a certificate depends on the trust in the certifying key by the
2267 recipient of the certificate.

2268 The key to be certified must be loaded before TPM_CertifyKey is called.

2269 The determination to use the TPM_CERTIFY_INFO or TPM_CERTIFY_INFO2 on the output is
2270 based on which PCRs and what localities the certified key is restricted to. A key to be
2271 certified that does not have locality restrictions and which uses no PCRs greater than PCR
2272 #15 will cause this command to return and sign a TPM_CERTIFY_INFO structure, which
2273 provides compatibility with V1.1 TPMs.

2274 When this command is run to certify all other keys (those that use PCR #16 or higher, as
2275 well as those limited by locality in any way), it will return and sign a TPM_CERTIFY_INFO2
2276 structure.

2277 TPM_CertifyKey does not support the case where (a) the certifying key requires a usage
2278 authorization to be provided but (b) the key-to-be-certified does not. In such cases,
2279 TPM_CertifyKey2 must be used. TPM_CertifyKey cannot be used to certify CMKs.

2280 If a command tag (in the parameter array) specifies only one authorisation session, then the
2281 TPM convention is that the first session listed is ignored (authDataUsage must be NEVER
2282 for this key) and the incoming session data is used for the second auth session in the list.
2283 In TPM_CertifyKey, the first session is the certifying key and the second session is the key-
2284 to-be-certified. In TPM_CertifyKey2, the first session is the key-to-be-certified and the
2285 second session is the certifying key.

2286 The key handles of both the certifying key and the key to be certified are not included in the
2287 HMAC protecting the command. This permits key handle virtualization (swapping of keys
2288 in and out of the TPM that results in different key handles while at the same time
2289 maintaining key identifiers of upper layer software). In environments where the interface to
2290 the TPM is accessible by other parties, the key handles not being protected allows an
2291 attacker to change the handle of the key to be certified. This can be avoided by processing
2292 this command within a transport session and making sure that antiReplay indeed contains
2293 a nonce.

2294 **End of informative comment.**

2295 **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_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.

2296 **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.

2297 **Actions**

- 2298 1. The TPM validates that the key pointed to by certHandle has a signature scheme of
2299 TPM_SS_RSASSAPKCS1v15_SHA1 or TPM_SS_RSASSAPKCS1v15_INFO
- 2300 2. Verify command and key AuthData values:
- 2301 a. If tag is TPM_TAG_RQU_AUTH2_COMMAND
- 2302 i. The TPM verifies the AuthData in certAuthHandle provides authorization to use
2303 the key pointed to by certHandle, return TPM_AUTHFAIL on error
- 2304 ii. The TPM verifies the AuthData in keyAuthHandle provides authorization to use
2305 the key pointed to by keyHandle, return TPM_AUTH2FAIL on error
- 2306 b. else if tag is TPM_TAG_RQU_AUTH1_COMMAND
- 2307 i. Verify that authDataUsage is TPM_AUTH_NEVER for the key referenced by
2308 certHandle, return TPM_AUTHFAIL on error.
- 2309 ii. The TPM verifies the AuthData in keyAuthHandle provides authorization to use
2310 the key pointed to by keyHandle, return TPM_AUTHFAIL on error
- 2311 c. else if tag is TPM_TAG_RQU_COMMAND
- 2312 i. Verify that authDataUsage is TPM_AUTH_NEVER for the key referenced by
2313 certHandle, return TPM_AUTHFAIL on error.

- 2314 ii. Verify that authDataUsage is TPM_AUTH_NEVER or TPM_AUTH_PRIV_USE_ONLY
2315 for the key referenced by keyHandle, return TPM_AUTHFAIL on error.
- 2316 3. If keyHandle -> payload is not TPM_PT_ASYM, return TPM_INVALID_KEYUSAGE.
- 2317 4. If the key pointed to by certHandle is an identity key (certHandle -> keyUsage is
2318 TPM_KEY_IDENTITY)
- 2319 a. If keyHandle -> keyFlags -> migratable is TRUE return TPM_MIGRATEFAIL
- 2320 5. Validate that certHandle -> keyUsage is TPM_KEY_SIGN, TPM_KEY_IDENTITY or
2321 TPM_KEY_LEGACY, if not return TPM_INVALID_KEYUSAGE
- 2322 6. Validate that keyHandle -> keyUsage is TPM_KEY_SIGN, TPM_KEY_STORAGE,
2323 TPM_KEY_IDENTITY, TPM_KEY_BIND or TPM_KEY_LEGACY, if not return
2324 TPM_INVALID_KEYUSAGE
- 2325 7. If keyHandle -> digestAtRelease requires the use of PCRs 16 or higher to calculate or if
2326 keyHandle -> localityAtRelease is not 0x1F
- 2327 a. Set V1 to 1.2
- 2328 8. Else
- 2329 a. Set V1 to 1.1
- 2330 9. If keyHandle -> pcrInfoSize is not 0
- 2331 a. If keyHandle -> keyFlags has pcrIgnoredOnRead set to FALSE
- 2332 i. Create a digestAtRelease according to the specified TPM_STCLEAR_DATA -> PCR
2333 registers and compare to keyHandle -> digestAtRelease and if a mismatch return
2334 TPM_WRONGPCRVAL
- 2335 ii. If specified validate any locality requests on error TPM_BAD_LOCALITY
- 2336 b. If V1 is 1.1
- 2337 i. Create C1 a TPM_CERTIFY_INFO structure
- 2338 ii. Fill in C1 with the information from the key pointed to by keyHandle
- 2339 iii. The TPM MUST set c1 -> pcrInfoSize to 44.
- 2340 iv. The TPM MUST set c1 -> pcrInfo to a TPM_PCR_INFO structure properly filled out
2341 using the information from keyHandle.
- 2342 v. The TPM MUST set c1 -> digestAtCreation to 20 bytes of 0x00.
- 2343 c. Else
- 2344 i. Create C1 a TPM_CERTIFY_INFO2 structure
- 2345 ii. Fill in C1 with the information from the key pointed to by keyHandle
- 2346 iii. Set C1 -> pcrInfoSize to the size of an appropriate TPM_PCR_INFO_SHORT
2347 structure.
- 2348 iv. Set C1 -> pcrInfo to a properly filled out TPM_PCR_INFO_SHORT structure, using
2349 the information from keyHandle.
- 2350 v. Set C1 -> migrationAuthoritySize to 0
- 2351 10. Else

- 2352 a. Create C1 a TPM_CERTIFY_INFO structure
- 2353 b. Fill in C1 with the information from the key pointed to by keyHandle
- 2354 c. The TPM MUST set c1 -> pcrInfoSize to 0
- 2355 11. Create TPM_DIGEST H1 which is the SHA-1 hash of keyHandle -> pubKey -> key. Note
2356 that <key> is the actual public modulus, and does not include any structure formatting.
- 2357 12. Set C1 -> pubKeyDigest to H1
- 2358 13. The TPM copies the antiReplay parameter to c1 -> data.
- 2359 14. The TPM sets certifyInfo to C1.
- 2360 15. The TPM creates m1, a message digest formed by taking the SHA-1 of c1.
- 2361 a. The TPM then computes a signature using certHandle -> sigScheme. The resulting
2362 signed blob is returned in outData.

2363 **13.9 TPM_CertifyKey2**

2364 **Start of informative comment:**

2365 This command is based on TPM_CertifyKey, but includes the ability to certify a Certifiable
2366 Migration Key (CMK), which requires extra input parameters.

2367 TPM_CertifyKey2 always produces a TPM_CERTIFY_INFO2 structure.

2368 TPM_CertifyKey2 does not support the case where (a) the key-to-be-certified requires a
2369 usage authorization to be provided but (b) the certifying key does not.

2370 If a command tag (in the parameter array) specifies only one authorisation session, then the
2371 TPM convention is that the first session listed is ignored (authDataUsage must be NEVER
2372 for this key) and the incoming session data is used for the second auth session in the list.
2373 In TPM_CertifyKey2, the first session is the key to be certified and the second session is the
2374 certifying key.

2375 The key handles of both the certifying key and the key to be certified are not included in the
2376 HMAC protecting the command. This permits key handle virtualization (swapping of keys
2377 in and out of the TPM that results in different key handles while at the same time
2378 maintaining key identifiers of upper layer software). In environments where the interface to
2379 the TPM is accessible by other parties, the key handles not being protected allows an
2380 attacker to change the handle of the key to be certified. This can be avoided by processing
2381 this command within a transport session and making sure that antiReplay indeed contains
2382 a nonce.

2383 **End of informative comment.**

2384 **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.

2385 **Outgoing Operands and Sizes**

Param		HMAC		Type	Name	Description
#	Sz	#	Sz			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CertifyKey2
4	<>	3S	<>	TPM_CERTIFY_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.

2386 **Actions**

- 2387 1. The TPM validates that the key pointed to by certHandle has a signature scheme of
2388 TPM_SS_RSASSAPKCS1v15_SHA1 or TPM_SS_RSASSAPKCS1v15_INFO
- 2389 2. Verify command and key AuthData values:
- 2390 a. If tag is TPM_TAG_RQU_AUTH2_COMMAND
- 2391 i. The TPM verifies the AuthData in keyAuthHandle provides authorization to use
2392 the key pointed to by keyHandle, return TPM_AUTHFAIL on error
- 2393 ii. The TPM verifies the AuthData in certAuthHandle provides authorization to use
2394 the key pointed to by certHandle, return TPM_AUTH2FAIL on error
- 2395 b. else if tag is TPM_TAG_RQU_AUTH1_COMMAND
- 2396 i. Verify that authDataUsage is TPM_AUTH_NEVER or TPM_AUTH_PRIV_USE_ONLY
2397 for the key referenced by keyHandle, return TPM_AUTHFAIL on error
- 2398 ii. The TPM verifies the AuthData in certAuthHandle provides authorization to use
2399 the key pointed to by certHandle, return TPM_AUTH2FAIL on error
- 2400 c. else if tag is TPM_TAG_RQU_COMMAND
- 2401 i. Verify that authDataUsage is TPM_AUTH_NEVER or TPM_AUTH_PRIV_USE_ONLY
2402 for the key referenced by keyHandle, return TPM_AUTHFAIL on error
- 2403 ii. Verify that authDataUsage is TPM_AUTH_NEVER for the key referenced by
2404 certHandle, return TPM_AUTHFAIL on error.

- 2405 3. If the key pointed to by certHandle is an identity key (certHandle -> keyUsage is
2406 TPM_KEY_IDENTITY)
- 2407 a. If keyHandle -> keyFlags -> migratable is TRUE and [keyHandle -> keyFlags->
2408 migrateAuthority is FALSE or (keyHandle -> payload != TPM_PT_MIGRATE_RESTRICTED
2409 and keyHandle -> payload != TPM_PT_MIGRATE_EXTERNAL)] return
2410 TPM_MIGRATEFAIL
- 2411 4. Validate that certHandle -> keyUsage is TPM_KEY_SIGNING, TPM_KEY_IDENTITY or
2412 TPM_KEY_LEGACY, if not return TPM_INVALID_KEYUSAGE
- 2413 5. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING, TPM_KEY_STORAGE,
2414 TPM_KEY_IDENTITY, TPM_KEY_BIND or TPM_KEY_LEGACY, if not return
2415 TPM_INVALID_KEYUSAGE
- 2416 6. The TPM SHALL create a c1 a TPM_CERTIFY_INFO2 structure from the key pointed to
2417 by keyHandle
- 2418 7. Create TPM_DIGEST H1 which is the SHA-1 hash of keyHandle -> pubKey -> key. Note
2419 that <key> is the actual public modulus, and does not include any structure formatting.
- 2420 8. Set C1 -> pubKeyDigest to H1
- 2421 9. Copy the antiReplay parameter to c1 -> data
- 2422 10. Copy other keyHandle parameters into C1
- 2423 11. If keyHandle -> payload == TPM_PT_MIGRATE_RESTRICTED or
2424 TPM_PT_MIGRATE_EXTERNAL
- 2425 a. create thisPubKey, a TPM_PUBKEY structure containing the public key, algorithm
2426 and parameters corresponding to keyHandle
- 2427 b. Verify that the migration authorization is valid for this key
- 2428 i. Create M2 a TPM_CMK_MIGAUTH structure
- 2429 ii. Set M2 -> msaDigest to migrationPubDigest
- 2430 iii. Set M2 -> pubkeyDigest to SHA-1[thisPubKey]
- 2431 iv. Verify that [keyHandle -> migrationAuth] == HMAC(M2) signed by using tpmProof
2432 as the secret and return error TPM_MA_SOURCE on mismatch
- 2433 c. Set C1 -> migrationAuthority = SHA-1(migrationPubDigest || keyHandle -> payload)
- 2434 d. if keyHandle -> payload == TPM_PT_MIGRATE_RESTRICTED
- 2435 i. Set C1 -> payloadType = TPM_PT_MIGRATE_RESTRICTED
- 2436 e. if keyHandle -> payload == TPM_PT_MIGRATE_EXTERNAL
- 2437 i. Set C1 -> payloadType = TPM_PT_MIGRATE_EXTERNAL
- 2438 12. Else
- 2439 a. set C1 -> migrationAuthority = NULL
- 2440 b. set C1 -> migrationAuthoritySize = 0
- 2441 c. Set C1 -> payloadType = TPM_PT_ASYM
- 2442 13. If keyHandle -> pcrInfoSize is not 0

- 2443 a. The TPM MUST set `c1 -> pcrInfoSize` to match the `pcrInfoSize` from the `keyHandle`
2444 key.
- 2445 b. The TPM MUST set `c1 -> pcrInfo` to match the `pcrInfo` from the `keyHandle` key
- 2446 c. If `keyHandle -> keyFlags` has `pcrIgnoredOnRead` set to `FALSE`
- 2447 i. Create a `digestAtRelease` according to the specified `TPM_STCLEAR_DATA -> PCR`
2448 registers and compare to `keyHandle -> digestAtRelease` and if a mismatch return
2449 `TPM_WRONGPCRVAL`
- 2450 ii. If specified validate any locality requests on error `TPM_BAD_LOCALITY`
- 2451 14.Else
- 2452 a. The TPM MUST set `c1 -> pcrInfoSize` to 0
- 2453 15.The TPM creates `m1`, a message digest formed by taking the SHA-1 of `c1`
- 2454 a. The TPM then computes a signature using `certHandle -> sigScheme`. The resulting
2455 signed blob is returned in `outData`

2456 **14. Endorsement Key Handling**2457 **Start of informative comment:**

2458 There are two create EK commands. The first matches the 1.1 functionality. The second
2459 provides the mechanism to enable revokeEK.

2460 The TPM and platform manufacturer decide on the inclusion or exclusion of the ability to
2461 execute revokeEK.

2462 The restriction to have the TPM generate the EK does not remove the manufacturing option
2463 to “squirt” the EK. During manufacturing, the TPM does not enforce all protections or
2464 requirements; hence, the restriction on only TPM generation of the EK is also not in force.

2465 **End of informative comment.**

- 2466 1. A TPM SHALL NOT install an EK unless generated on the TPM by execution of
2467 TPM_CreateEndorsementKeyPair or TPM_CreateRevocableEK

2468 **14.1 TPM_CreateEndorsementKeyPair**

2469 **Start of informative comment:**

2470 This command creates the TPM endorsement key. It returns a failure code if an
2471 endorsement key already exists.

2472 **End of informative comment.**

2473 **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

2474 **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

2475 **Actions**

- 2476 1. If an EK already exists, return TPM_DISABLED_CMD
- 2477 2. Validate the keyInfo parameters for the key description
- 2478 a. If the algorithm type is RSA the key length MUST be a minimum of 2048. For
2479 interoperability the key length SHOULD be 2048
- 2480 b. If the algorithm type is other than RSA the strength provided by the key MUST be
2481 comparable to RSA 2048
- 2482 c. The other parameters of keyInfo (signatureScheme etc.) are ignored.
- 2483 3. Create a key pair called the “endorsement key pair” using a TPM-protected capability.
2484 The type and size of key are that indicated by keyInfo
- 2485 4. Create checksum by performing SHA-1 on the concatenation of (PUBEK || antiReplay)
- 2486 5. Store the PRIVEK
- 2487 6. Create TPM_PERMANENT_DATA -> tpmDAASeed from the TPM RNG

- 2488 7. Create TPM_PERMANENT_DATA -> daaProof from the TPM RNG
- 2489 8. Create TPM_PERMANENT_DATA -> daaBlobKey from the TPM RNG
- 2490 9. Set TPM_PERMANENT_FLAGS -> CEKPUsed to TRUE
- 2491 10. Set TPM_PERMANENT_FLAGS -> enableRevokeEK to FALSE

2492 14.2 TPM_CreateRevocableEK

2493 **Start of informative comment:**

2494 This command creates the TPM endorsement key. It returns a failure code if an
2495 endorsement key already exists. The TPM vendor may have a separate mechanism to create
2496 the EK and “squirt” the value into the TPM.

2497 The input parameters specify whether the EK is capable of being reset, whether the
2498 AuthData value to reset the EK will be generated by the TPM, and the new AuthData value
2499 itself if it is not to be generated by the TPM. The output parameter is the new AuthData
2500 value that must be used when resetting the EK (if it is capable of being reset).

2501 The command TPM_RevokeTrust must be used to reset an EK (if it is capable of being
2502 reset).

2503 Owner authorisation is unsuitable for authorizing resetting of an EK: someone with
2504 Physical Presence can remove a genuine Owner, install a new Owner, and revoke the EK.
2505 The genuine Owner can reinstall, but the platform will have lost its original attestation and
2506 may not be trusted by challengers. Therefore if a password is to be used to revoke an EK, it
2507 must be a separate password, given to the genuine Owner.

2508 In v1.2 an OEM has extra choices when creating EKs.

2509 a) An OEM could manufacture all of its TPMs with enableRevokeEK==TRUE.

2510 If the OEM has tracked the EKreset passwords for these TPMs, the OEM can give the
2511 passwords to customers. The customers can use the passwords as supplied, change the
2512 passwords, or clear the EKs and create new EKs with new passwords.

2513 If EKreset passwords are random values, the OEM can discard those values and not give
2514 them to customers. There is then a low probability (statistically zero) chance of a local DOS
2515 attack to reset the EK by guessing the password. The chance of a remote DOS attack is zero
2516 because Physical Presence must also be asserted to use TPM_RevokeTrust.

2517 b) An OEM could manufacture some of its TPMs with enableRevokeEK==FALSE. Then the
2518 EK can never be revoked, and the chance of even a local DOS attack on the EK is
2519 eliminated.

2520 **End of informative comment.**

2521 This is an optional command

2522 **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

2523 **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

2524 **Actions**

- 2525 1. If an EK already exists, return TPM_DISABLED_CMD
- 2526 2. Perform the actions of TPM_CreateEndorsementKeyPair, if any errors return with error
- 2527 3. Set TPM_PERMANENT_FLAGS -> enableRevokeEK to TRUE
- 2528 a. If generateReset is TRUE then
- 2529 i. Set TPM_PERMANENT_DATA -> EKreset to the next value from the TPM RNG
- 2530 b. Else
- 2531 i. Set TPM_PERMANENT_DATA -> EKreset to inputEKreset
- 2532 4. Return PUBEK, checksum and Ekreset
- 2533 5. The outputEKreset AuthData is sent in the clear. There is no uniqueness on the TPM
- 2534 available to actually perform encryption or use an encrypted channel. The assumption is
- 2535 that this operation is occurring in a controlled environment and sending the value in the
- 2536 clear is acceptable.

2537 **14.3 TPM_RevokeTrust**

2538 **Start of informative comment:**

2539 This command clears the EK and sets the TPM back to a pure default state. The generation
2540 of the AuthData value occurs during the generation of the EK. It is the responsibility of the
2541 EK generator to properly protect and disseminate the RevokeTrust AuthData.

2542 **End of informative comment.**

2543 This is an optional command

2544 **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

2545 **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

2546 **Actions**

- 2547 1. The TPM MUST validate that TPM_PERMANENT_FLAGS -> enableRevokeEK is TRUE,
2548 return TPM_PERMANENTEK on error
- 2549 2. The TPM MUST validate that the EKReset matches TPM_PERMANENT_DATA -> EKReset
2550 return TPM_AUTHFAIL on error.
- 2551 3. Ensure that physical presence is being asserted
- 2552 4. Perform the actions of TPM_OwnerClear (excepting the command authentication)
- 2553 a. NV items with the pubInfo -> nvIndex D value set MUST be deleted. This changes the
2554 TPM_OwnerClear handling of the same NV areas
- 2555 b. Set TPM_PERMANENT_FLAGS -> nvLocked to FALSE
- 2556 5. Invalidate TPM_PERMANENT_DATA -> tpmDAASeed
- 2557 6. Invalidate TPM_PERMANENT_DATA -> daaProof
- 2558 7. Invalidate TPM_PERMANENT_DATA -> daaBlobKey
- 2559 8. Invalidate the EK and any internal state associated with the EK

2560 **14.4 TPM_ReadPubek**2561 **Start of informative comment:**

2562 Return the endorsement key public portion. This value should have controls placed upon
2563 access, as it is a privacy sensitive value.

2564 The readPubek flag is set to FALSE by TPM_TakeOwnership and set to TRUE by
2565 TPM_OwnerClear, thus mirroring if a TPM Owner is present.

2566 **End of informative comment.**2567 **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

2568 **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

2569 **Description**

2570 This command returns the PUBEK.

2571 **Actions**

2572 The TPM_ReadPubek command SHALL

2573 1. If TPM_PERMANENT_FLAGS -> readPubek is FALSE return TPM_DISABLED_CMD

2574 2. If no EK is present the TPM MUST return TPM_NO_ENDORSEMENT

2575 3. Create checksum by performing SHA-1 on the concatenation of (pubEndorsementKey ||
2576 antiReplay).

2577 4. Export the PUBEK and checksum.

2578 **14.5 TPM_OwnerReadInternalPub**

2579 **Start of informative comment:**

2580 A TPM Owner authorized command that returns the public portion of the EK or SRK.

2581 The keyHandle parameter is included in the incoming session authorization to prevent
2582 alteration of the value, causing a different key to be read. Unlike most key handles, which
2583 can be mapped by higher layer software, this key handle has only two fixed values.

2584 **End of informative comment.**

2585 **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.

2586 **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.

2587 **Actions**

- 2588 1. Validate the parameters and TPM Owner AuthData for this command
- 2589 2. If keyHandle is TPM_KH_EK
- 2590 a. Set publicPortion to PUBEK

- 2591 3. Else If keyHandle is TPM_KH_SRK
- 2592 a. Set publicPortion to the TPM_PUBKEY of the SRK
- 2593 4. Else return TPM_BAD_PARAMETER
- 2594 5. Export the public key of the referenced key

2595 **15. Identity Creation and Activation**

2596 **15.1 TPM_MakeIdentity**

2597 **Start of informative comment:**

2598 Generate a new Attestation Identity Key (AIK).

2599 labelPrivCADigest identifies the privacy CA that the owner expects to be the target CA for
2600 the AIK. The selection is not enforced by the TPM. It is advisory only. It is included
2601 because the TSS cannot be trusted to send the AIK to the correct privacy CA. The privacy
2602 CA can use this parameter to validate that it is the target privacy CA and label intended by
2603 the TPM owner at the time the key was created. The label can be used to indicate an
2604 application purpose.

2605 **End of informative comment.**

2606 **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	continueSrkJession	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.

2607 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal:TPM_ORD_MakeIdentity.
4	<>	3S	<>	TPM_KEY	idKey	The newly created identity key. MAY be TPM_KEY12
5	4	4S	4	UINT32	identityBindingSize	The used size of the output area for identityBinding
6	<>	5S	<>	BYTE[]	identityBinding	Signature of TPM_IDENTITY_CONTENTS using idKey.private.
7	20	2H2	20	TPM_NONCE	srkNonceEven	Even nonce newly generated by TPM.
		3H2	20	TPM_NONCE	srknonceOdd	Nonce generated by system associated with srkAuthHandle
8	1	4H2	1	BOOL	continueSrkJession	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.

2608 **Description**

2609 The public key of the new TPM identity SHALL be identityPubKey. The private key of the
2610 new TPM identity SHALL be tpm_signature_key.

2611 **Properties of the new identity**

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

2612
2613 This capability also generates a TPM_KEY containing the tpm_signature_key.

2614 If identityPubKey is stored on a platform it SHALL exist only in storage to which access is
2615 controlled and is available to authorized entities.

2616 **Actions**

2617 A Trusted Platform Module that receives a valid TPM_MakeIdentity command SHALL do the
2618 following:

- 2619 1. Validate the idKeyParams parameters for the key description
 - 2620 a. If the algorithm type is RSA the key length MUST be a minimum of 2048. For
2621 interoperability the key length SHOULD be 2048

- 2622 b. If the algorithm type is other than RSA the strength provided by the key MUST be
2623 comparable to RSA 2048
- 2624 c. If the TPM is not designed to create a key of the requested type, return the error code
2625 TPM_BAD_KEY_PROPERTY
- 2626 d. If TPM_PERMANENT_FLAGS -> FIPS is TRUE then
- 2627 i. If authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS
- 2628 2. Use authHandle to verify that the Owner authorized all TPM_MakeIdentity input
2629 parameters.
- 2630 3. Use srkAuthHandle to verify that the SRK owner authorized all TPM_MakeIdentity input
2631 parameters.
- 2632 4. Verify that idKeyParams -> keyUsage is TPM_KEY_IDENTITY. If it is not, return
2633 TPM_INVALID_KEYUSAGE
- 2634 5. Verify that idKeyParams -> keyFlags -> migratable is FALSE. If it is not, return
2635 TPM_INVALID_KEYUSAGE
- 2636 6. Create a1 by decrypting identityAuth according to the ADIP indicated by authHandle.
- 2637 7. Set continueAuthSession and continueSRKSession to FALSE.
- 2638 8. Determine the structure version
- 2639 a. If idKeyParams -> tag is TPM_TAG_KEY12
- 2640 i. Set V1 to 2
- 2641 ii. Create idKey a TPM_KEY12 structure using idKeyParams as the default values for
2642 the structure
- 2643 b. If idKeyParams -> ver is 1.1
- 2644 i. Set V1 to 1
- 2645 ii. Create idKey a TPM_KEY structure using idKeyParams as the default values for
2646 the structure
- 2647 9. Set the digestAtCreation values for pcrInfo
- 2648 a. For TPM_PCR_INFO_LONG include the locality of the current command
- 2649 10. Create an asymmetric key pair (identityPubKey and tpm_signature_key) using a TPM-
2650 protected capability, in accordance with the algorithm specified in idKeyParams
- 2651 11. Ensure that the AuthData information in A1 is properly stored in the idKey as
2652 usageAuth.
- 2653 12. Attach identityPubKey and tpm_signature_key to idKey
- 2654 13. Set idKey -> migrationAuth to TPM_PERMANENT_DATA-> tpmProof
- 2655 14. Ensure that all TPM_PAYLOAD_TYPE structures identify this key as TPM_PT_ASYM
- 2656 15. Encrypt the private portion of idKey using the SRK as the parent key
- 2657 16. Create a TPM_IDENTITY_CONTENTS structure named idContents using
2658 labelPrivCADigest and the information from idKey

2659 17. Sign idContents using tpm_signature_key and TPM_SS_RSASSAPKCS1v15_SHA1. Store
2660 the result in identityBinding.

2661 **15.2 TPM_ActivateIdentity**

2662 **Start of informative comment:**

2663 The purpose of TPM_ActivateIdentity is to twofold. The first purpose is to obtain assurance
2664 that the credential in the TPM_SYM_CA_ATTESTATION is for this TPM. The second purpose
2665 is to obtain the session key used to encrypt the TPM_IDENTITY_CREDENTIAL.

2666 This is an extension to the 1.1 functionality of TPM_ActivateIdentity. The blob sent to from
2667 the CA can be in the 1.1 format or the 1.2 format. The TPM determines the type from the
2668 size or version information in the blob.

2669 TPM_ActivateIdentity checks that the symmetric session key corresponds to a TPM-identity
2670 before releasing that session key.

2671 Only the Owner of the TPM has the privilege of activating a TPM identity. The Owner is
2672 required to authorize the TPM_ActivateIdentity command. The owner may authorize the
2673 command using either the TPM_OIAP or TPM_OSAP authorization protocols.

2674 The creator of the ActivateIdentity package can specify if any PCR values are to be checked
2675 before releasing the session key.

2676 **End of informative comment.**

2677 **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.

2678 **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.

2679 **Description**

- 2680 1. The command TPM_ActivateIdentity activates a TPM identity created using the command
2681 TPM_MakeIdentity.
- 2682 2. The command assumes the availability of the private key associated with the identity.
2683 The command will verify the association between the keys during the process.
- 2684 3. The command will decrypt the input blob and extract the session key and verify the
2685 connection between the public and private keys. The input blob can be in 1.1 or 1.2
2686 format.

2687 **Actions**

- 2688 A Trusted Platform Module that receives a valid TPM_ActivateIdentity command SHALL do
2689 the following:
- 2690 1. Using the authHandle field, validate the owner's AuthData to execute the command and
2691 all of the incoming parameters.
- 2692 2. Using the idKeyAuthHandle, validate the AuthData to execute command and all of the
2693 incoming parameters
- 2694 3. Validate that the idKey is the public key of a valid TPM identity by checking that
2695 idKeyHandle -> keyUsage is TPM_KEY_IDENTITY. Return TPM_BAD_PARAMETER on
2696 mismatch
- 2697 4. Create H1 the digest of a TPM_PUBKEY derived from idKey
- 2698 5. Decrypt blob creating B1 using PRIVEK as the decryption key

- 2699 6. Determine the type and version of B1
- 2700 a. If B1 -> tag is TPM_TAG_EK_BLOB then
- 2701 i. B1 is a TPM_EK_BLOB
- 2702 b. Else
- 2703 i. B1 is a TPM_ASYM_CA_CONTENTS. As there is no tag for this structure it is
- 2704 possible for the TPM to make a mistake here but other sections of the structure
- 2705 undergo validation
- 2706 7. If B1 is a version 1.1 TPM_ASYM_CA_CONTENTS then
- 2707 a. Compare H1 to B1 -> idDigest on mismatch return TPM_BAD_PARAMETER
- 2708 b. Set K1 to B1 -> sessionKey
- 2709 8. If B1 is a TPM_EK_BLOB then
- 2710 a. Validate that B1 -> ekType is TPM_EK_TYPE_ACTIVATE, return TPM_BAD_TYPE if
- 2711 not.
- 2712 b. Assign A1 as a TPM_EK_BLOB_ACTIVATE structure from B1 -> blob
- 2713 c. Compare H1 to A1 -> idDigest on mismatch return TPM_BAD_PARAMETER
- 2714 d. If A1 -> pcrSelection is not NULL
- 2715 i. Compute a composite hash C1 using the PCR selection A1 -> pcrSelection
- 2716 ii. Compare C1 to A1 -> pcrInfo->digestAtRelease and return TPM_WRONGPCRVAL
- 2717 on a mismatch
- 2718 iii. If A1 -> pcrInfo specifies a locality ensure that the appropriate locality has been
- 2719 asserted, return TPM_BAD_LOCALITY on error
- 2720 e. Set K1 to A1 -> symmetricKey
- 2721 9. Return K1

2722 **16. Integrity Collection and Reporting**2723 **Start of informative comment:**

2724 This section deals with what commands have direct access to the PCR

2725 **End of informative comment.**2726 1. The TPM SHALL only allow the following commands to alter the value of
2727 TPM_STCLEAR_DATA -> PCR

2728 a. TPM_Extend

2729 b. TPM_SHA1CompleteExtend

2730 c. TPM_Startup

2731 d. TPM_PCR_Reset

2732 **16.1 TPM_Extend**

2733 **Start of informative comment:**

2734 This adds a new measurement to a PCR

2735 **End of informative comment.**

2736 **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.

2737 **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.

2738 **Descriptions**

2739 Add a measurement value to a PCR

2740 **Actions**

2741 1. Map L1 to TPM_STANY_FLAGS -> localityModifier

2742 2. Map P1 to TPM_PERMANENT_DATA -> pcrAttrib [pcrNum]. pcrExtendLocal

2743 3. If, for the value of L1, the corresponding bit is not set in the bit map P1, return
2744 TPM_BAD_LOCALITY

2745 4. Create c1 by concatenating (TPM_STCLEAR_DATA -> PCR[pcrNum] || inDigest). This
2746 takes the current PCR value and concatenates the inDigest parameter.

2747 5. Create h1 by performing a SHA-1 digest of c1.

2748 6. Store h1 to TPM_STCLEAR_DATA -> PCR[pcrNum]

2749 7. If TPM_PERMANENT_FLAGS -> disable is TRUE or TPM_STCLEAR_FLAGS -> deactivated
2750 is TRUE

2751 a. Set outDigest to 20 bytes of 0x00

- 2752 8. Else
- 2753 a. Set outDigest to h1

2754 **16.2 TPM_PCRRead**

2755 **Start of informative comment:**

2756 The TPM_PCRRead operation provides non-cryptographic reporting of the contents of a
2757 named PCR.

2758 **End of informative comment.**

2759 **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

2760 **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

2761 **Description**

2762 The TPM_PCRRead operation returns the current contents of the named register to the
2763 caller.

2764 **Actions**

2765 1. Set outDigest to TPM_STCLEAR_DATA -> PCR[pcrIndex]

2766 2. Return TPM_SUCCESS

2767

2768 **16.3 TPM_Quote**2769 **Start of informative comment:**

2770 The TPM_Quote operation provides cryptographic reporting of PCR values. A loaded key is
2771 required for operation. TPM_Quote uses a key to sign a statement that names the current
2772 value of a chosen PCR and externally supplied data (which may be a nonce supplied by a
2773 Challenger).

2774 The term "ExternalData" is used because an important use of TPM_Quote is to provide a
2775 digital signature on arbitrary data, where the signature includes the PCR values of the
2776 platform at time of signing. Hence the "ExternalData" is not just for anti-replay purposes,
2777 although it is (of course) used for that purpose in an integrity challenge.

2778 **End of informative comment.**2779 **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.

2780 **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.

2781 **Actions**

- 2782 1. The TPM MUST validate the AuthData to use the key pointed to by keyHandle.
- 2783 2. Validate that keyHandle -> sigScheme is TPM_SS_RSASSAPKCS1v15_SHA1 or
2784 TPM_SS_RSASSAPKCS1v15_INFO, if not return TPM_INAPPROPRIATE_SIG.
- 2785 3. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING, TPM_KEY_IDENTITY, or
2786 TPM_KEY_LEGACY, if not return TPM_INVALID_KEYUSAGE
- 2787 4. Validate targetPCR
- 2788 a. targetPCR is a valid TPM_PCR_SELECTION structure
- 2789 b. On errors return TPM_INVALID_PCR_INFO
- 2790 5. Create H1 a SHA-1 hash of a TPM_PCR_COMPOSITE using the TPM_STCLEAR_DATA ->
2791 PCR indicated by targetPCR -> pcrSelect
- 2792 6. Create Q1 a TPM_QUOTE_INFO structure
- 2793 a. Set Q1 -> version to 1.1.0.0
- 2794 b. Set Q1 -> fixed to "QUOT"
- 2795 c. Set Q1 -> digestValue to H1
- 2796 d. Set Q1 -> externalData to externalData
- 2797 7. Sign SHA-1 hash of Q1 using keyHandle as the signature key
- 2798 8. Return the signature in sig

2799 **16.4 TPM_PCR_Reset**2800 **Start of informative comment:**

2801 For PCR with the pcrReset attribute set to TRUE, this command resets the PCR back to the
2802 default value, this mimics the actions of TPM_Init. The PCR may have restrictions as to
2803 which locality can perform the reset operation.

2804 Sending a null pcrSelection results in an error is due to the requirement that the command
2805 actually do something. If pcrSelection is null there are no PCR to reset and the command
2806 would then do nothing.

2807 For PCR that are resettable, the presence of a Trusted Operating System (TOS) can change
2808 the behavior of TPM_PCR_Reset. The following pseudo code shows how the behavior
2809 changes

2810 At TPM_Startup

2811 If TPM_PCR_ATTRIBUTES->pcrReset is FALSE

2812 Set PCR to 0x00...00

2813 Else

2814 Set PCR to 0xFF...FF

2815 At TPM_PCR_Reset

2816 If TPM_PCR_ATTRIBUTES->pcrReset is TRUE

2817 If TOSPresent

2818 Set PCR to 0x00...00

2819 Else

2820 Set PCR to 0xFF...FF

2821 Else

2822 Return error

2823 The above pseudocode is for example only, for the details of a specific platform, the reader
2824 must review the platform specific specification. The purpose of the above pseudocode is to
2825 show that both pcrReset and the TOSPresent bit control the value in use to when the PCR
2826 resets.

2827 **End of informative comment.**

2828 **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

2829 **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

2830 **Descriptions**

2831 This command resets PCR values back to the default value. The command MUST validate
 2832 that all PCR registers that are selected are available to be reset before resetting any PCR.
 2833 This command MUST either reset all selected PCR registers or none of the PCR registers.

2834 **Actions**

- 2835 1. Validate that pcrSelection is valid
 - 2836 a. is a valid TPM_PCR_SELECTION structure
 - 2837 b. pcrSelection -> pcrSelect is non-zero
 - 2838 c. On errors return TPM_INVALID_PCR_INFO
- 2839 2. Map L1 to TPM_STANY_FLAGS -> localityModifier
- 2840 3. For each PCR selected perform the following
 - 2841 a. If TPM_PERMANENT_DATA -> pcrAttrib[pcrIndex].pcrReset is FALSE, return
 2842 TPM_NOTRESETABLE
 - 2843 b. If, for the value L1, the corresponding bit is clear in the bit map
 2844 TPM_PERMANENT_DATA -> pcrAttrib[pcrIndex].pcrResetLocal, return TPM_NOTLOCAL
- 2845 4. For each PCR selected perform the following
 - 2846 a. The PCR MAY only reset to 0x00...00 or 0xFF...FF
 - 2847 b. The logic to determine which value to use MUST be described by a platform specific
 2848 specification

2849 **16.5 TPM_Quote2**2850 **Start of informative comment:**

2851 The TPM_Quote2 operation provides cryptographic reporting of PCR values. A loaded key is
2852 required for operation. TPM_Quote2 uses a key to sign a statement that names the current
2853 value of a chosen PCR and externally supplied data (which may be a nonce supplied by a
2854 Challenger).

2855 The term "externalData" is used because an important use of TPM_Quote2 is to provide a
2856 digital signature on arbitrary data, where the signature includes the PCR values of the
2857 platform at time of signing. Hence the "externalData" is not just for anti-replay purposes,
2858 although it is (of course) used for that purpose in an integrity challenge.

2859 TPM_Quote2 differs from TPM_Quote in that TPM_Quote2 uses TPM_PCR_INFO_SHORT to
2860 hold information relative to the PCR registers. TPM_PCR_INFO_SHORT includes locality
2861 information to provide the requestor a more complete view of the current platform
2862 configuration.

2863 **End of informative comment.**2864 **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.

2865 **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.

2866 **Actions**

- 2867 1. The TPM MUST validate the AuthData to use the key pointed to by keyHandle.
- 2868 2. Validate that keyHandle -> sigScheme is TPM_SS_RSASSAPKCS1v15_SHA1 or
2869 TPM_SS_RSASSAPKCS1v15_INFO, if not return TPM_INAPPROPRIATE_SIG.
- 2870 3. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING, TPM_KEY_IDENTITY, or
2871 TPM_KEY_LEGACY, if not return TPM_INVALID_KEYUSAGE
- 2872 4. Validate targetPCR is a valid TPM_PCR_SELECTION structure, on errors return
2873 TPM_INVALID_PCR_INFO
- 2874 5. Create H1 a SHA-1 hash of a TPM_PCR_COMPOSITE using the TPM_STCLEAR_DATA ->
2875 PCR[] indicated by targetPCR -> pcrSelect
- 2876 6. Create S1 a TPM_PCR_INFO_SHORT
- 2877 a. Set S1->pcrSelection to targetPCR
- 2878 b. Set S1->localityAtRelease to TPM_STANY_DATA -> localityModifier
- 2879 c. Set S1->digestAtRelease to H1
- 2880 7. Create Q1 a TPM_QUOTE_INFO2 structure
- 2881 a. Set Q1 -> fixed to "QUT2"
- 2882 b. Set Q1 -> infoShort to S1
- 2883 c. Set Q1 -> externalData to externalData
- 2884 8. If addVersion is TRUE
- 2885 a. Concatenate to Q1 a TPM_CAP_VERSION_INFO structure

- 2886 b. Set the output parameters for versionInfo
- 2887 9. Else
- 2888 a. Set versionInfoSize to 0
- 2889 b. Return no bytes in versionInfo
- 2890 10. Sign a SHA-1 hash of Q1 using keyHandle as the signature key
- 2891 11. Return the signature in sig

2892 **17. Changing AuthData**

2893 **17.1 TPM_ChangeAuth**

2894 **Start of informative comment:**

2895 The TPM_ChangeAuth command allows the owner of an entity to change the AuthData for
2896 the entity.

2897 This command cannot invalidate the old entity. Therefore, the authorization change is only
2898 effective if the application can guarantee that the old entity can be securely destroyed. If
2899 not, two valid entities will exist, one with the old and one with the new authorization secret.

2900 If this command is delegated, the delegated party can expand its key use privileges. That
2901 party can create a copy of the key with known authorization, and it can then use the key
2902 without any ordinal restrictions.

2903 TPM_ChangeAuth requires the encryption of one parameter (“NewAuth”). For the sake of
2904 uniformity with other commands that require the encryption of more than one parameter,
2905 the parameters used for used encryption are generated from the authLastNonceEven
2906 (created during the OSAP session), nonceOdd, and the session shared secret.

2907 The parameter list to this command must always include two authorization sessions,
2908 regardless of the state of authDataUsage for the respective keys.

2909 **End of informative comment.**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: 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.
----	----	--	--------------	------------	--

2910 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation. See section 4.3.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: 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

2911 **Description**

- 2912 1. The parentAuthHandle session type MUST be TPM_PID_OSAP.
- 2913 2. In this capability, the SRK cannot be accessed as entityType TPM_ET_KEY, since the
- 2914 SRK is not wrapped by a parent key.

2915 **Actions**

- 2916 1. Verify that entityType is one of TPM_ET_DATA, TPM_ET_KEY and return the error
- 2917 TPM_WRONG_ENTITYTYPE if not.
- 2918 2. Verify that parentAuthHandle session type is TPM_PID_OSAP return TPM_BAD_MODE
- 2919 on error
- 2920 3. Verify that entityAuthHandle session type is TPM_PID_OIAP return TPM_BAD_MODE on
- 2921 error
- 2922 4. If protocolID is not TPM_PID_ADCP, the TPM MUST return TPM_BAD_PARAMETER.
- 2923 5. The encData parameter MUST be the encData field from either the TPM_STORED_DATA
- 2924 or TPM_KEY structures.
- 2925 6. Create decryptAuth by decrypting newAuth according to the ADIP indicated by
- 2926 parentHandle.
- 2927 7. The TPM MUST validate the command using the AuthData in the parentAuth parameter
- 2928 8. Validate that parentHandle -> keyUsage is TPM_KEY_STORAGE, if not return
- 2929 TPM_INVALID_KEYUSAGE

- 2930 9. After parameter validation, the TPM creates b1 by decrypting encData using the key
2931 pointed to by parentHandle.
- 2932 10. The TPM MUST validate that b1 is a valid TPM structure, either a
2933 TPM_STORE_ASYMKEY or a TPM_SEALED_DATA
- 2934 a. Check the tag, length and authValue for match, return TPM_INVALID_STRUCTURE
2935 on any mismatch
- 2936 11. The TPM replaces the AuthData for b1 with decryptAuth created above.
- 2937 12. The TPM encrypts b1 using the appropriate mechanism for the type using the
2938 parentKeyHandle to provide the key information.
- 2939 13. The TPM MUST enforce the destruction of both the parentAuthHandle and
2940 entityAuthHandle sessions.

2941 **17.2 TPM_ChangeAuthOwner**

2942 **Start of informative comment:**

2943 The TPM_ChangeAuthOwner command allows the owner of an entity to change the
2944 AuthData for the TPM Owner or the SRK.

2945 This command requires authorization from the current TPM Owner to execute.

2946 TPM's targeted for an environment (e.g. a server) with long lasting sessions should not
2947 invalidate all sessions.

2948 **End of informative comment.**

2949 **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.

2950 **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

2951 **Actions**

- 2952 1. The TPM MUST validate the command using the AuthData in the ownerAuth parameter
- 2953 2. The ownerAuthHandle session type MUST be TPM_PID_OSAP
- 2954 3. If protocolID is not TPM_PID_ADCP, the TPM MUST return TPM_BAD_PARAMETER.
- 2955 4. Verify that entityType is either TPM_ET_OWNER or TPM_ET_SRK, and return the error
2956 TPM_WRONG_ENTITYTYPE if not.
- 2957 5. Create decryptAuth by decrypting newAuth according to the ADIP indicated by
2958 ownerAuthHandle.
- 2959 6. The TPM MUST enforce the destruction of the ownerAuthHandle session upon
2960 completion of this command (successful or unsuccessful). This includes setting
2961 continueAuthSession to FALSE
- 2962 7. Set the AuthData for the indicated entity to decryptAuth
- 2963 8. The TPM MUST invalidate all owner authorized OSAP and DSAP sessions, active or
2964 saved.
- 2965 9. The TPM MAY invalidate all sessions, active or saved

2966 **18. Authorization Sessions**

2967 **18.1 TPM_OIAP**

2968 **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.

2969 **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.

2970 **Actions**

- 2971 1. The TPM_OIAP command allows the creation of an authorization session handle and the
2972 tracking of the handle by the TPM. The TPM generates the handle and nonce.
- 2973 2. The TPM has an internal limit as to the number of handles that may be open at one
2974 time, so the request for a new handle may fail if there is insufficient space available.
- 2975 3. Internally the TPM will do the following:
 - 2976 a. TPM allocates space to save handle, protocol identification, both nonces and any
2977 other information the TPM needs to manage the session.
 - 2978 b. TPM generates authHandle and nonceEven, returns these to caller
- 2979 4. On each subsequent use of the OIAP session the TPM MUST generate a new nonceEven
2980 value.
- 2981 5. When TPM_OIAP is wrapped in an encrypted transport session, no input or output
2982 parameters are encrypted.

2983

2984 **18.1.1 Actions to validate an OIAP session**2985 **Start of informative comment:**

2986 This section describes the authorization-related actions of a TPM when it receives a
2987 command that has been authorized with the OIAP protocol.

2988 Many commands use OIAP authorization. The following description is therefore necessarily
2989 abstract.

2990 **End of informative comment.**2991 **Actions**

2992 The TPM MUST perform the following operations:

- 2993 1. The TPM MUST verify that the authorization session handle (H, say) referenced in the
2994 command points to a valid session. If it does not, the TPM returns the error code
2995 TPM_INVALID_AUTHHANDLE
- 2996 2. The TPM SHALL retrieve the latest version of the caller's nonce (nonceOdd) and
2997 continueAuthSession flag from the input parameter list, and store it in internal TPM
2998 memory with the authSession 'H'.
- 2999 3. The TPM SHALL retrieve the latest version of the TPM's nonce stored with the
3000 authorization session H (authLastNonceEven) computed during the previously executed
3001 command.
- 3002 4. The TPM MUST retrieve the secret AuthData (SecretE, say) of the target entity. The
3003 entity and its secret must have been previously loaded into the TPM.
 - 3004 a. If the command using the OIAP session requires owner authorization
 - 3005 i. If TPM_STCLEAR_DATA -> ownerReference is TPM_KH_OWNER, the secret
3006 AuthData is TPM_PERMANENT_DATA -> ownerAuth
 - 3007 ii. If TPM_STCLEAR_DATA -> ownerReference is pointing to a delegate row
 - 3008 (1) Set R1 a row index to TPM_STCLEAR_DATA -> ownerReference
 - 3009 (2) Set D1 a TPM_DELEGATE_TABLE_ROW to TPM_PERMANENT_DATA ->
3010 delegateTable -> delRow[R1]
 - 3011 (3) Set the secret AuthData to D1 -> authValue
 - 3012 (4) Validate the TPM_DELEGATE_PUBLIC D1 -> pub
 - 3013 (a) Validate D1 -> pub -> permissions based on the command ordinal
 - 3014 (b) Validate D1 -> pub -> pcrInfo based on the PCR values
 - 3015 5. The TPM SHALL perform a HMAC calculation using the entity secret data, ordinal, input
3016 command parameters and authorization parameters per Part 1 Object-Independent
3017 Authorization Protocol.
 - 3018 6. The TPM SHALL compare HM to the AuthData value received in the input parameters. If
3019 they are different, the TPM returns the error code TPM_AUTHFAIL if the authorization
3020 session is the first session of a command, or TPM_AUTH2FAIL if the authorization

- 3021 session is the second session of a command. Otherwise, the TPM executes the command
3022 which (for this example) produces an output that requires authentication.
- 3023 7. The TPM SHALL generate a nonce (nonceEven).
- 3024 8. The TPM creates an HMAC digest to authenticate the return code, return values and
3025 authorization parameters to the same entity secret per Part 1 Object-Independent
3026 Authorization Protocol.
- 3027 9. The TPM returns the return code, output parameters, authorization parameters and
3028 authorization session digest.
- 3029 10. If the output continueUse flag is FALSE, then the TPM SHALL terminate the session.
3030 Future references to H will return an error.

3031 **18.2 TPM_OSAP**3032 **Start of informative comment:**

3033 The TPM_OSAP command creates the authorization session handle, the shared secret and
3034 generates nonceEven and nonceEvenOSAP.

3035 **End of informative comment.**3036 **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.

3037 **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.

3038 **Description**

- 3039 1. The TPM_OSAP command allows the creation of an authorization session handle and the
3040 tracking of the handle by the TPM. The TPM generates the handle, nonceEven and
3041 nonceEvenOSAP.
- 3042 2. The TPM has an internal limit on the number of handles that may be open at one time,
3043 so the request for a new handle may fail if there is insufficient space available.
- 3044 3. The TPM_OSAP allows the binding of an authorization to a specific entity. This allows
3045 the caller to continue to send in AuthData for each command but not have to request
3046 the information or cache the actual AuthData.
- 3047 4. When TPM_OSAP is wrapped in an encrypted transport session, no input or output
3048 parameters are encrypted.
- 3049 5. If the owner pointer is pointing to a delegate row, the TPM internally MUST treat the
3050 OSAP session as a DSAP session

3051 6. TPM_ET_SRK or TPM_ET_KEYHANDLE with a value of TPM_KH_SRK MUST specify the
3052 SRK.

3053 7. If the entity is tied to PCR values, the PCR's are not validated during the TPM_OSAP
3054 ordinal session creation. The PCR's are validated when the OSAP session is used.

3055 **Actions**

3056 1. The TPM creates S1 a storage area that keeps track of the information associated with
3057 the authorization.

3058 2. S1 MUST track the following information

3059 a. Protocol identification (i.e. TPM_PID_OSAP)

3060 b. nonceEven

3061 i. Initialized to the next value from the TPM RNG

3062 c. shared secret

3063 d. ADIP encryption scheme from TPM_ENTITY_TYPE entityType

3064 e. Any other internal TPM state the TPM needs to manage the session

3065 3. The TPM MUST create and MAY track the following information

3066 a. nonceEvenOSAP

3067 i. Initialized to the next value from the TPM RNG

3068 4. The TPM calculates the shared secret using an HMAC calculation. The key for the HMAC
3069 calculation is the secret AuthData assigned to the key handle identified by entityType.
3070 The input to the HMAC calculation is the concatenation of nonces nonceEvenOSAP and
3071 nonceOddOSAP. The output of the HMAC calculation is the shared secret which is saved
3072 in the authorization area associated with authHandle

3073 5. Check if the ADIP encryption scheme specified by entityType is supported, if not return
3074 TPM_INAPPROPRIATE_ENC.

3075 6. If entityType = TPM_ET_KEYHANDLE

3076 a. The entity to authorize is a key held in the TPM. entityType contains the keyHandle
3077 that holds the key.

3078 b. If entityType is TPM_KH_OPERATOR return TPM_BAD_HANDLE

3079 7. else if entityType = TPM_ET_OWNER

3080 a. This value indicates that the entity is the TPM owner. entityType is ignored

3081 b. The HMAC key is the secret pointed to by ownerReference (owner secret or delegated
3082 secret)

3083 8. else if entityType = TPM_ET_SRK

3084 a. The entity to authorize is the SRK. entityType is ignored.

3085 9. else if entityType = TPM_ET_COUNTER

3086 a. The entity is a monotonic counter, entityType contains the counter handle

3087 10. else if entityType = TPM_ET_NV

- 3088 a. The entity is a NV index, entityValue contains the NV index
- 3089 11.else return TPM_BAD_PARAMETER
- 3090 12.On each subsequent use of the OSAP session the TPM MUST generate a new nonce
3091 value.
- 3092 13.The TPM MUST ensure that OSAP shared secret is only available while the OSAP session
3093 is valid.
- 3094 14.The session MUST terminate upon any of the following conditions:
- 3095 a. The command that uses the session returns an error
- 3096 b. The resource is evicted from the TPM or otherwise invalidated
- 3097 c. The session is used in any command for which the shared secret is used to encrypt
3098 an input parameter (TPM_ENCAUTH)
- 3099 d. The TPM Owner is cleared
- 3100 e. TPM_ChangeAuthOwner is executed and this session is attached to the owner
3101 authorization
- 3102 f. The session explicitly terminated with continueAuth, TPM_Reset or
3103 TPM_FlushSpecific
- 3104 g. All OSAP sessions associated with the delegation table MUST be invalidated when
3105 any of the following commands execute:
- 3106 i. TPM_Delegate_Manage
- 3107 ii. TPM_Delegate_CreateOwnerDelegation with Increment==TRUE
- 3108 iii. TPM_Delegate_LoadOwnerDelegation

3109

3110 **18.2.1 Actions to validate an OSAP session**

3111 **Start of informative comment:**

3112 This section describes the authorization-related actions of a TPM when it receives a
3113 command that has been authorized with the OSAP protocol.

3114 Many commands use OSAP authorization. The following description is therefore necessarily
3115 abstract.

3116 **End of informative comment**

3117 **Actions**

- 3118 1. On reception of a command with ordinal C1 that uses an authorization session, the TPM
3119 SHALL perform the following actions:
- 3120 2. The TPM MUST have been able to retrieve the shared secret (Shared, say) of the target
3121 entity when the authorization session was established with TPM_OSAP. The entity and
3122 its secret must have been previously loaded into the TPM.
- 3123 3. The TPM MUST verify that the authorization session handle (H, say) referenced in the
3124 command points to a valid session. If it does not, the TPM returns the error code
3125 TPM_INVALID_AUTHHANDLE.
- 3126 4. The TPM MUST calculate the HMAC (HM1, say) of the command parameters according
3127 to Part 1 Object-Specific Authorization Protocol.
- 3128 5. The TPM SHALL compare HM1 to the AuthData value received in the command. If they
3129 are different, the TPM returns the error code TPM_AUTHFAIL if the authorization session
3130 is the first session of a command, or TPM_AUTH2FAIL if the authorization session is the
3131 second session of a command., the TPM executes command C1 which produces an
3132 output (O, say) that requires authentication and uses a particular return code (RC, say).
- 3133 6. The TPM SHALL generate the latest version of the even nonce (nonceEven).
- 3134 7. The TPM MUST calculate the HMAC (HM2) of the return parameters according to section
3135 Part 1 Object-Specific Authorization Protocol.
- 3136 8. The TPM returns HM2 in the parameter list.
- 3137 9. The TPM SHALL retrieve the continue flag from the received command. If the flag is
3138 FALSE, the TPM SHALL terminate the session and destroy the thread associated with
3139 handle H.
- 3140 10.If the shared secret was used to provide confidentiality for data in the received
3141 command, the TPM SHALL terminate the session and destroy the thread associated with
3142 handle H.
- 3143 11.Each time that access to an entity (e.g., key) is authorized using OSAP, the TPM MUST
 - 3144 a. ensure that the OSAP shared secret is that derived from the entity using TPM_OSAP
 - 3145 b. validate the PCR values if specified for the entity

- 3146
3147
3148
- i. The TPM SHOULD validate the PCR values before using the shared secret to validate the command parameters. This prevents a dictionary attack on the shared secret when the PCR values are invalid for the entity.

3149 **18.3 TPM_DSAP**

3150 **Start of informative comment:**

3151 The TPM_DSAP command creates the authorization session handle using a delegated
3152 AuthData value passed into the command as an encrypted blob or from the internal
3153 delegation table. It can be used to start an authorization session for a user key or the
3154 owner.

3155 Identically to TPM_OSAP, it generates a shared secret and generates nonceEven and
3156 nonceEvenOSAP.

3157 **End of informative comment.**

3158 **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 entityType 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

3159 **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.

3160 **Description**

3161 1. The TPM_DSAP command allows the creation of an authorization session handle and the
3162 tracking of the handle by the TPM. The TPM generates the handle, nonceEven and
3163 nonceEvenOSAP.

- 3164 2. The TPM has an internal limit on the number of handles that may be open at one time,
3165 so the request for a new handle may fail if there is insufficient space available.
- 3166 3. The TPM_DSAP allows the binding of a delegated authorization to a specific entity. This
3167 allows the caller to continue to send in AuthData for each command but not have to
3168 request the information or cache the actual AuthData.
- 3169 4. Each ordinal that uses the DSAP session MUST validate that TPM_PERMANENT_DATA -
3170 > restrictDelegate does not restrict delegation, based on keyHandle -> keyUsage and
3171 keyHandle -> keyFlags, return TPM_INVALID_KEYUSAGE on error.
- 3172 5. On each subsequent use of the DSAP session the TPM MUST generate a new nonce
3173 value and check if the ordinal to be executed has delegation to execute. The TPM MUST
3174 ensure that the DSAP shared secret is only available while the DSAP session is valid.
- 3175 6. When TPM_DSAP is wrapped in an encrypted transport session
- 3176 a. For input the only parameter encrypted is entityValue
- 3177 b. For output no parameters are encrypted
- 3178 7. The DSAP session MUST terminate under any of the following conditions
- 3179 a. The command that uses the session returns an error
- 3180 b. If attached to a key, when the key is evicted from the TPM or otherwise invalidated
- 3181 c. The session is used in any command for which the shared secret is used to encrypt
3182 an input parameter (TPM_ENCAUTH)
- 3183 d. The TPM Owner is cleared
- 3184 e. TPM_ChangeAuthOwner is executed and this session is attached to the owner
3185 authorization
- 3186 f. The session explicitly terminated with continueAuth, TPM_Reset or
3187 TPM_FlushSpecific
- 3188 g. All DSAP sessions MUST be invalidated when any of the following commands
3189 execute:
- 3190 i. TPM_Delegate_CreateOwnerDelegation
- 3191 ii. When Increment is TRUE
- 3192 iii. TPM_Delegate_LoadOwnerDelegation
- 3193 iv. TPM_Delegate_Manage
- 3194 entityType = TPM_ET_DEL_OWNER_BLOB
- 3195 The entityValue parameter contains an owner delegation blob structure.
- 3196 entityType = TPM_ET_DEL_ROW
- 3197 The entityValue parameter contains a row number in the nv Delegation table which
3198 should be used for the AuthData value.
- 3199 entityType = TPM_DEL_KEY_BLOB
- 3200 The entityValue parameter contains a key delegation blob structure.

3201 Actions

- 3202 1. If entityType == TPM_ET_DEL_OWNER_BLOB
- 3203 a. Map entityValue to B1 a TPM_DELEGATE_OWNER_BLOB
- 3204 b. Validate that B1 is a valid TPM_DELEGATE_OWNER_BLOB, return
3205 TPM_WRONG_ENTITYTYPE on error
- 3206 c. Locate B1 -> pub -> familyID in the TPM_FAMILY_TABLE and set familyRow to
3207 indicate row, return TPM_BADINDEX if not found
- 3208 d. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]
- 3209 e. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
- 3210 f. Verify that B1->verificationCount equals FR -> verificationCount.
- 3211 g. Validate the integrity of the blob
- 3212 i. Copy B1 -> integrityDigest to H2
- 3213 ii. Set B1 -> integrityDigest to all zeros
- 3214 iii. Create H3 the HMAC of B1 using tpmProof as the secret
- 3215 iv. Compare H2 to H3 return TPM_AUTHFAIL on mismatch
- 3216 h. Create S1 a TPM_DELEGATE_SENSITIVE by decrypting B1 -> sensitiveArea using
3217 TPM_DELEGATE_KEY
- 3218 i. Validate S1 values
- 3219 i. S1 -> tag is TPM_TAG_DELEGATE_SENSITIVE
- 3220 ii. Return TPM_BAD_DELEGATE on error
- 3221 j. Set A1 to S1 -> authValue
- 3222 2. Else if entityType == TPM_ET_DEL_ROW
- 3223 a. Verify that entityValue points to a valid row in the delegation table.
- 3224 b. Set D1 to the delegation information in the row.
- 3225 c. Set A1 to D1->authValue.
- 3226 d. Locate D1 -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate that
3227 row, return TPM_BADINDEX if not found
- 3228 e. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]
- 3229 f. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
- 3230 g. Verify that D1->verificationCount equals FR -> verificationCount.
- 3231 3. Else if entityType == TPM_ET_DEL_KEY_BLOB
- 3232 a. Map entityValue to K1 a TPM_DELEGATE_KEY_BLOB
- 3233 b. Validate that K1 is a valid TPM_DELEGATE_KEY_BLOB, return
3234 TPM_WRONG_ENTITYTYPE on error
- 3235 c. Locate K1 -> pub -> familyID in the TPM_FAMILY_TABLE and set familyRow to
3236 indicate that row, return TPM_BADINDEX if not found
- 3237 d. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]

- 3238 e. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
- 3239 f. Verify that K1 -> pub -> verificationCount equals FR -> verificationCount.
- 3240 g. Validate the integrity of the blob
- 3241 i. Copy K1 -> integrityDigest to H2
- 3242 ii. Set K1 -> integrityDigest to all zeros
- 3243 iii. Create H3 the HMAC of K1 using tpmProof as the secret
- 3244 iv. Compare H2 to H3 return TPM_AUTHFAIL on mismatch
- 3245 h. Validate that K1 -> pubKeyDigest identifies keyHandle, return TPM_KEYNOTFOUND
- 3246 on error
- 3247 i. Create S1 a TPM_DELEGATE_SENSITIVE by decrypting K1 -> sensitiveArea using
- 3248 TPM_DELEGATE_KEY
- 3249 j. Validate S1 values
- 3250 i. S1 -> tag is TPM_TAG_DELEGATE_SENSITIVE
- 3251 ii. Return TPM_BAD_DELEGATE on error
- 3252 k. Set A1 to S1 -> authValue
- 3253 4. Else return TPM_BAD_PARAMETER
- 3254 5. Generate a new authorization session handle and reserve space to save protocol
- 3255 identification, shared secret, pcrInfo, both nonces, ADIP encryption scheme, delegated
- 3256 permission bits and any other information the TPM needs to manage the session.
- 3257 6. Read two new values from the RNG to generate nonceEven and nonceEvenOSAP.
- 3258 7. The TPM calculates the shared secret using an HMAC calculation. The key for the HMAC
- 3259 calculation is A1. The input to the HMAC calculation is the concatenation of nonces
- 3260 nonceEvenOSAP and nonceOddOSAP. The output of the HMAC calculation is the shared
- 3261 secret which is saved in the authorization area associated with authHandle.

3262 **18.4 TPM_SetOwnerPointer**

3263 **Start of informative comment:**

3264 This command will set a reference to which secret the TPM will use when executing an
3265 owner secret related OIAP or OSAP session.

3266 This command should only be used to provide an owner delegation function for legacy code
3267 that does not itself support delegation. Normally, TPM_STCLEAR_DATA->ownerReference
3268 points to TPM_KH_OWNER, indicating that OIAP and OSAP sessions should use the owner
3269 authorization. This command allows ownerReference to point to an index in the delegation
3270 table, indicating that OIAP and OSAP sessions should use the delegation authorization.

3271 In use, a TSS supporting delegation would create and load the owner delegation and set the
3272 owner pointer to that delegation. From then on, a legacy TSS application would use its OIAP
3273 and OSAP sessions with the delegated owner authorization.

3274 Since this command is not authorized, the ownerReference is open to DoS attacks.
3275 Applications can attempt to recover from a failing owner authorization by resetting
3276 ownerReference to an appropriate value.

3277 **End of informative comment.**

3278 **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

3279 **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

3280 **Actions**

3281 1. Map TPM_STCLEAR_DATA to V1

3282 2. If entityType = TPM_ET_DEL_ROW

3283 a. This value indicates that the entity is a delegate row. entityValue is a delegate index
3284 in the delegation table.

- 3285 b. Validate that entityValue points to a legal row within the delegate table stored within
3286 the TPM. If not return TPM_BADINDEX
- 3287 i. Set D1 to the delegation information in the row.
- 3288 c. Locate D1 -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate that
3289 row, return TPM_BADINDEX if not found.
- 3290 d. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]
- 3291 e. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
- 3292 f. Verify that B1->verificationCount equals FR -> verificationCount.
- 3293 g. The TPM sets V1-> ownerReference to entityValue
- 3294 h. Return TPM_SUCCESS
- 3295 3. else if entityType = TPM_ET_OWNER
- 3296 a. This value indicates that the entity is the TPM owner. entityValue is ignored.
- 3297 b. The TPM sets V1-> ownerReference to TPM_KH_OWNER
- 3298 c. Return TPM_SUCCESS
- 3299 4. Return TPM_BAD_PARAMETER

3300 19. Delegation Commands

3301 19.1 TPM_Delegate_Manage

3302 **Start of informative comment:**

3303 TPM_Delegate_Manage is the fundamental process for managing the Family tables,
3304 including enabling/disabling Delegation for a selected Family. Normally
3305 TPM_Delegate_Manage must be executed at least once (to create Family tables for a
3306 particular family) before any other type of Delegation command in that family can succeed.

3307 TPM_Delegate_Manage is authorized by the TPM Owner if an Owner is installed, because
3308 changing a table is a privileged Owner operation. If no Owner is installed,
3309 TPM_Delegate_Manage requires no privilege to execute. This does not disenfranchise an
3310 Owner, since there is no Owner, and simplifies loading of tables during platform
3311 manufacture or on first-boot. Burn-out of TPM non-volatile storage by inappropriate use is
3312 mitigated by the TPM's normal limits on NV-writes in the absence of an Owner. Tables can
3313 be locked after loading, to prevent subsequent tampering, and only unlocked by the Owner,
3314 his delegate, or the act of removing the Owner (even if there is no Owner).

3315 TPM_Delegate_Manage command is customized by opCode:

3316 (1) TPM_FAMILY_ENABLE enables/disables use of a family and all the rows of the delegate
3317 table belonging to that family,

3318 (2) TPM_FAMILY_ADMIN can be used to prevent further management of the Tables until an
3319 Owner is installed, or until the Owner is removed from the TPM. (Note that the Physical
3320 Presence command TPM_ForceClear always enables further management, even if
3321 TPM_ForceClear is used when no Owner is installed.)

3322 (3) TPM_FAMILY_CREATE creates a new family. Sessions are invalidated even in this case
3323 because the lastFamilyID could wrap.

3324 (4) TPM_FAMILY_INVALIDATE invalidates an existing family. The TPM_SELFTEST_FAILED
3325 error code is returned because the familyRow has already been validated earlier. Failure
3326 here indicates a malfunction of the TPM.

3327 **End of informative comment.**

3328 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.

3329 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.

3330 Action

3331 1. If opCode != TPM_FAMILY_CREATE

3332 a. Locate familyID in the TPM_FAMILY_TABLE and set familyRow to indicate row,
3333 return TPM_BADINDEX if not found3334 b. Set FR, a TPM_FAMILY_TABLE_ENTRY, to TPM_FAMILY_TABLE. famTableRow
3335 [familyRow]

3336 2. If tag = TPM_TAG_RQU_AUTH1_COMMAND

3337 a. Validate the command and parameters using ownerAuth, return TPM_AUTHFAIL on
3338 error

- 3339 b. If the command is delegated (authHandle session type is TPM_PID_DSAP or through
3340 ownerReference delegation)
- 3341 i. If opCode = TPM_FAMILY_CREATE
- 3342 (1) The TPM MUST ignore familyID
- 3343 ii. Else
- 3344 (1) Verify that the familyID associated with authHandle matches the familyID
3345 parameter, return TPM_DELEGATE_FAMILY on error
- 3346 3. Else
- 3347 a. If TPM_PERMANENT_DATA -> ownerAuth is valid, return TPM_AUTHFAIL
- 3348 b. If opCode != TPM_FAMILY_CREATE and FR -> flags ->
3349 TPM_DELEGATE_ADMIN_LOCK is TRUE, return TPM_DELEGATE_LOCK
- 3350 c. Validate max NV writes without an owner
- 3351 i. Set NV1 to TPM_PERMANENT_DATA -> noOwnerNVWrite
- 3352 ii. Increment NV1 by 1
- 3353 iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITES
- 3354 iv. Set TPM_PERMANENT_DATA -> noOwnerNVWrite to NV1
- 3355 4. The TPM invalidates sessions
- 3356 a. MUST invalidate all DSAP sessions
- 3357 b. MUST invalidate all OSAP sessions associated with the delegation table
- 3358 c. MUST set TPM_STCLEAR_DATA -> ownerReference to TPM_KH_OWNER
- 3359 d. MAY invalidate any other session
- 3360 5. If opCode == TPM_FAMILY_CREATE
- 3361 a. Validate that sufficient space exists within the TPM to store an additional family and
3362 map F2 to the newly allocated space.
- 3363 b. Validate that opData is a TPM_FAMILY_LABEL
- 3364 i. If opDataSize != sizeof(TPM_FAMILY_LABEL) return TPM_BAD_PARAM_SIZE
- 3365 c. Map F2 to a TPM_FAMILY_TABLE_ENTRY
- 3366 i. Set F2 -> tag to TPM_TAG_FAMILY_TABLE_ENTRY
- 3367 ii. Set F2 -> familyLabel to opData
- 3368 d. Increment TPM_PERMANENT_DATA -> lastFamilyID by 1
- 3369 e. Set F2 -> familyID = TPM_PERMANENT_DATA -> lastFamilyID
- 3370 f. Set F2 -> verificationCount = 1
- 3371 g. Set F2 -> flags -> TPM_FAMFLAG_ENABLED to FALSE
- 3372 h. Set F2 -> flags -> TPM_DELEGATE_ADMIN_LOCK to FALSE
- 3373 i. Set retDataSize = 4

- 3374 j. Set retData = F2 -> familyID
- 3375 k. Return TPM_SUCCESS
- 3376 6. If authHandle is of type DSAP then continueAuthSession MUST set to FALSE
- 3377 7. If opCode == TPM_FAMILY_ADMIN
 - 3378 a. Validate that opDataSize == 1, and that opData is a Boolean value.
 - 3379 b. Set (FR -> flags -> TPM_DELEGATE_ADMIN_LOCK) = opData
 - 3380 c. Set retDataSize = 0
 - 3381 d. Return TPM_SUCCESS
- 3382 8. else If opCode == TPM_FAMILY_ENABLE
 - 3383 a. Validate that opDataSize == 1, and that opData is a Boolean value.
 - 3384 b. Set FR -> flags-> TPM_FAMFLAG_ENABLED = opData
 - 3385 c. Set retDataSize = 0
 - 3386 d. Return TPM_SUCCESS
- 3387 9. else If opCode == TPM_FAMILY_INVALIDATE
 - 3388 a. Invalidate all data associated with familyRow
 - 3389 i. All data is all information pointed to by FR
 - 3390 ii. return TPM_SELFTEST_FAILED on failure
 - 3391 b. Set retDataSize = 0
 - 3392 c. Return TPM_SUCCESS
- 3393 10. Else return TPM_BAD_PARAMETER

3394 **19.2 TPM_Delegate_CreateKeyDelegation**

3395 **Start of informative comment:**

3396 This command delegates privilege to use a key by creating a blob that can be used by
3397 TPM_DSAP.

3398 There is no check for appropriateness of the key's key usage against the key permission
3399 settings. If the key usage is incorrect, this command succeeds, but the delegated command
3400 will fail.

3401 These blobs CANNOT be used as input data for TPM_LoadOwnerDelegation because the
3402 internal TPM delegate table can store owner delegations only.

3403 (TPM_Delegate_CreateOwnerDelegation must be used to delegate Owner privilege.)

3404 **End of informative comment**

3405 **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

3406

3407 **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

3408 **Description**

3409 1. The use restrictions that may be present on the key pointed to by keyHandle are not
 3410 enforced for this command. Stated another way, TPM_CreateKeyDelegation is not a use
 3411 of the key.

3412 **Action**

- 3413 1. Verify AuthData for the command and parameters using privAuth
- 3414 2. Locate publicInfo -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate
 3415 row, return TPM_BADINDEX if not found
- 3416 3. If the key authentication is in fact a delegation, then the TPM SHALL validate the
 3417 command and parameters using Delegation authorisation, then
- 3418 a. Validate that authHandle -> familyID equals publicInfo -> familyID return
 3419 TPM_DELEGATE_FAMILY on error
- 3420 b. If TPM_FAMILY_TABLE.famTableRow[authHandle -> familyID] -> flags ->
 3421 TPM_FAMFLAG_ENABLED is FALSE, return error TPM_DISABLED_CMD.
- 3422 c. Verify that the delegation bits in publicInfo do not grant more permissions then
 3423 currently delegated. Otherwise return error TPM_AUTHFAIL
- 3424 4. Check that publicInfo -> delegateType is TPM_DEL_KEY_BITS
- 3425 5. Verify that authHandle indicates an OSAP or DSAP session return
 3426 TPM_INVALID_AUTHHANDLE on error
- 3427 6. Create a1 by decrypting delAuth according to the ADIP indicated by authHandle.
- 3428 7. Create h1 the SHA-1 of TPM_STORE_PUBKEY structure of the key pointed to by
 3429 keyHandle
- 3430 8. Create M1 a TPM_DELEGATE_SENSITIVE structure
- 3431 a. Set M1 -> tag to TPM_TAG_DELEGATE_SENSITIVE

- 3432 b. Set M1 -> authValue to a1
- 3433 c. The TPM MAY add additional information of a sensitive nature relative to the
- 3434 delegation
- 3435 9. Create M2 the encryption of M1 using TPM_DELEGATE_KEY
- 3436 10. Create P1 a TPM_DELEGATE_KEY_BLOB
- 3437 a. Set P1 -> tag to TPM_TAG_DELG_KEY_BLOB
- 3438 b. Set P1 -> pubKeyDigest to H1
- 3439 c. Set P1 -> pub to PublicInfo
- 3440 d. Set P1 -> pub -> verificationCount to familyRow -> verificationCount
- 3441 e. Set P1 -> integrityDigest to all zeros
- 3442 f. The TPM sets additionalArea and additionalAreaSize appropriate for this TPM. The
- 3443 information MAY include symmetric IV, symmetric mode of encryption and other data
- 3444 that allows the TPM to process the blob in the future.
- 3445 g. Set P1 -> sensitiveSize to the size of M2
- 3446 h. Set P1 -> sensitiveArea to M2
- 3447 11. Calculate H2 the HMAC of P1 using tpmProof as the secret
- 3448 12. Set P1 -> integrityDigest to H2
- 3449 13. Ignore continueAuthSession on input set continueAuthSession to FALSE on output
- 3450 14. Return P1 as blob

3451 **19.3 TPM_Delegate_CreateOwnerDelegation**3452 **Start of informative comment:**

3453 TPM_Delegate_CreateOwnerDelegation delegates the Owner's privilege to use a set of
3454 command ordinals, by creating a blob. Such blobs can be used as input data for TPM_DSAP
3455 or TPM_Delegate_LoadOwnerDelegation.

3456 TPM_Delegate_CreateOwnerDelegation includes the ability to void all existing delegations
3457 (by incrementing the verification count) before creating the new delegation. This ensures
3458 that the new delegation will be the only delegation that can operate at Owner privilege in
3459 this family. This new delegation could be used to enable a security monitor (a local separate
3460 entity, or remote separate entity, or local host entity) to reinitialize a family and perhaps
3461 perform external verification of delegation settings. Normally the ordinals for a delegated
3462 security monitor would include TPM_Delegate_CreateOwnerDelegation (this command) in
3463 order to permit the monitor to create further delegations, and
3464 TPM_Delegate_UpdateVerification to reactivate some previously voided delegations.

3465 If the verification count is incremented and the new delegation does not delegate any
3466 privileges (to any ordinals) at all, or uses an authorisation value that is then discarded, this
3467 family's delegations are all void and delegation must be managed using actual Owner
3468 authorisation.

3469 (TPM_Delegate_CreateKeyDelegation must be used to delegate privilege to use a key.)

3470 **End of informative comment.**3471 **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

3472 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	TPM_ORD_Delegate_CreateOwnerDelegation
4	4	3S	4	UINT32	blobSize	The length of the returned blob
5	<>	4S	<>	TPM_DELEGATE_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

3473 **Action**

- 3474 1. The TPM SHALL authenticate the command using TPM Owner authentication. Return
3475 TPM_AUTHFAIL on failure.
- 3476 2. Locate publicInfo -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate
3477 the row return TPM_BADINDEX if not found
- 3478 a. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]
- 3479 3. If the TPM Owner authentication is in fact a delegation
- 3480 a. Validate that authHandle -> familyID equals publicInfo -> familyID return
3481 TPM_DELEGATE_FAMILY on error
- 3482 b. If FR -> flags -> TPM_FAMFLAG_ENABLED is FALSE, return error
3483 TPM_DISABLED_CMD.
- 3484 c. Verify that the delegation bits in publicInfo do not grant more permissions then
3485 currently delegated. Otherwise, return error TPM_AUTHFAIL.
- 3486 4. Check that publicInfo -> delegateType is TPM_DEL_OWNER_BITS
- 3487 5. Verify that authHandle indicates an OSAP or DSAP session return
3488 TPM_INVALID_AUTHHANDLE on error
- 3489 6. If increment == TRUE
- 3490 a. Increment FR -> verificationCount
- 3491 b. Set TPM_STCLEAR_DATA-> ownerReference to TPM_KH_OWNER
- 3492 c. The TPM invalidates sessions
- 3493 i. MUST invalidate all DSAP sessions
- 3494 ii. MUST invalidate all OSAP sessions associated with the delegation table

- 3495 iii. MAY invalidate any other session
- 3496 7. Create a1 by decrypting delAuth according to the ADIP indicated by authHandle.
- 3497 8. Create M1 a TPM_DELEGATE_SENSITIVE structure
- 3498 a. Set M1 -> tag to TPM_TAG_DELEGATE_SENSITIVE
- 3499 b. Set M1 -> authValue to a1
- 3500 c. Set other M1 fields as determined by the TPM vendor
- 3501 9. Create M2 the encryption of M1 using TPM_DELEGATE_KEY
- 3502 10. Create B1 a TPM_DELEGATE_OWNER_BLOB
- 3503 a. Set B1 -> tag to TPM_TAG_DELG_OWNER_BLOB
- 3504 b. Set B1 -> pub to publicInfo
- 3505 c. Set B1 -> sensitiveSize to the size of M2
- 3506 d. Set B1 -> sensitiveArea to M2
- 3507 e. Set B1 -> integrityDigest to all zeros
- 3508 f. Set B1 -> pub -> verificationCount to FR -> verificationCount
- 3509 11. The TPM sets additionalArea and additionalAreaSize appropriate for this TPM. The
- 3510 information MAY include symmetric IV, symmetric mode of encryption and other data
- 3511 that allows the TPM to process the blob in the future.
- 3512 12. Create H1 the HMAC of B1 using tpmProof as the secret
- 3513 13. Set B1 -> integrityDigest to H1
- 3514 14. Ignore continueAuthSession on input set continueAuthSession to FALSE on output
- 3515 15. Return B1 as blob

3516 **19.4 TPM_Delegate_LoadOwnerDelegation**

3517 **Start of informative comment:**

3518 This command loads a delegate table row blob into a non-volatile delegate table row.
3519 TPM_Delegate_LoadOwnerDelegation can be used during manufacturing or on first boot
3520 (when no Owner is installed), or after an Owner is installed. If an Owner is installed,
3521 TPM_Delegate_LoadOwnerDelegation requires Owner authorisation, and sensitive
3522 information must be encrypted.

3523 Burn-out of TPM non-volatile storage by inappropriate use is mitigated by the TPM’s normal
3524 limits on NV-writes in the absence of an Owner. Tables can be locked after loading using
3525 TPM_Delegate_Manage, to prevent subsequent tampering.

3526 A management system outside the TPM is expected to manage the delegate table rows
3527 stored on the TPM, and can overwrite any previously stored data.

3528 This command cannot be used to load key delegation blobs into the TPM

3529 **End of informative comment.**

3530 **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 row to be written
5	4	4S	4	UINT32	blobSize	The size of the delegate blob
6	<>	5S	<>	TPM_DELEGATE_OWNER_BLOB	blob	Delegation information, including encrypted portions as appropriate
7	4			TPM_AUTHHANDLE	authHandle	The authorization 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

3531 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation
		2S	4	TPM_COMMAND_CODE	ordinal	TPM_ORD_Delegate_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.

3532 **Actions**

- 3533 1. Map blob to D1 a TPM_DELEGATE_OWNER_BLOB.
- 3534 a. Validate that D1 -> tag == TPM_TAG_DELEGATE_OWNER_BLOB
- 3535 2. Locate D1 -> pub -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate
3536 row, return TPM_BADINDEX if not found
- 3537 3. Set FR to TPM_FAMILY_TABLE -> famTableRow[familyRow]
- 3538 4. If TPM Owner is installed
- 3539 a. Validate the command and parameters using TPM Owner authentication, return
3540 TPM_AUTHFAIL on error
- 3541 b. If the command is delegated (authHandle session type is TPM_PID_DSAP or through
3542 ownerReference delegation), verify that D1 -> pub -> familyID matches authHandle ->
3543 familyID, on error return TPM_DELEGATE_FAMILY
- 3544 5. Else
- 3545 a. If tag is not TPM_TAG_RQU_COMMAND, return TPM_AUTHFAIL
- 3546 b. If FR -> flags -> TPM_DELEGATE_ADMIN_LOCK is TRUE return
3547 TPM_DELEGATE_LOCK
- 3548 c. Validate max NV writes without an owner
- 3549 i. Set NV1 to PD -> noOwnerNVWrite
- 3550 ii. Increment NV1 by 1
- 3551 iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITES
- 3552 iv. Set PD -> noOwnerNVWrite to NV1
- 3553 6. If FR -> flags -> TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
- 3554 7. If TPM Owner is installed, validate the integrity of the blob
- 3555 a. Copy D1 -> integrityDigest to H2
- 3556 b. Set D1 -> integrityDigest to all zeros

- 3557 c. Create H3 the HMAC of D1 using tpmProof as the secret
- 3558 d. Compare H2 to H3, return TPM_AUTHFAIL on mismatch
- 3559 8. If TPM Owner is installed, create S1 a TPM_DELEGATE_SENSITIVE area by decrypting
3560 D1 -> sensitiveArea using TPM_DELEGATE_KEY. Otherwise set S1 = D1 -> sensitiveArea
- 3561 9. Validate S1
- 3562 a. Validate that S1 -> tag == TPM_TAG_DELEGATE_SENSITIVE, return
3563 TPM_INVALID_STRUCTURE on error
- 3564 10. Validate that index is a valid value for delegateTable, return TPM_BADINDEX on error
- 3565 11. The TPM invalidates sessions
- 3566 a. MUST invalidate all DSAP sessions
- 3567 b. MUST invalidate all OSAP sessions associated with the delegation table
- 3568 c. MAY invalidate any other session
- 3569 12. Copy data to the delegate table row
- 3570 a. Copy the TPM_DELEGATE_PUBLIC from D1 -> pub to TPM_DELEGATE_TABLE ->
3571 delRow[index] -> pub.
- 3572 b. Copy the TPM_SECRET from S1 -> authValue to TPM_DELEGATE_TABLE ->
3573 delRow[index] -> authValue.
- 3574 c. Set TPM_STCLEAR_DATA-> ownerReference to TPM_KH_OWNER
- 3575 d. If authHandle is of type DSAP then continueAuthSession MUST set to FALSE
- 3576 13. Return TPM_SUCCESS

3577 **19.5 TPM_Delegate_ReadTable**3578 **Start of informative comment:**

3579 This command reads from the TPM the public contents of the family and delegate tables
3580 that are stored on the TPM. Such data is required during external verification of tables.

3581 There are no restrictions on the execution of this command; anyone can read this
3582 information regardless of the state of the PCRs, regardless of whether they know any
3583 specific AuthData value and regardless of whether or not the enable and admin bits are set
3584 one way or the other.

3585 **End of informative comment.**3586 **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

3587 **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

3588 **Actions**

- 3589 1. Set familyTableSize to the number of valid families on the TPM times
3590 sizeof(TPM_FAMILY_TABLE_ELEMENT).
- 3591 2. Copy the valid entries in the internal family table to the output array familyTable
- 3592 3. Set delegateTableSize to the number of valid delegate table entries on the TPM times
3593 (sizeof(TPM_DELEGATE_PUBLIC) + 4).
- 3594 4. For each valid entry
- 3595 a. Write the TPM_DELEGATE_INDEX to delegateTable
- 3596 b. Copy the TPM_DELEGATE_PUBLIC to delegateTable

3597 5. Return TPM_SUCCESS

3598 **19.6 TPM_Delegate_UpdateVerification**3599 **Start of informative comment:**

3600 TPM_UpdateVerification sets the verificationCount in an entity (a blob or a delegation row)
3601 to the current family value, in order that the delegations represented by that entity will
3602 continue to be accepted by the TPM.

3603 **End of informative comment.**3604 **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.

3605 **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.

3606 **Actions**

- 3607 1. Verify the TPM Owner, directly or indirectly through delegation, authorizes the command
3608 and parameters, on error return TPM_AUTHFAIL
- 3609 2. Determine the type of inputData (TPM_DELEGATE_TABLE_ROW or
3610 TPM_DELEGATE_OWNER_BLOB or TPM_DELEGATE_KEY_BLOB) and map D1 to that
3611 structure
- 3612 a. Mapping to TPM_DELEGATE_TABLE_ROW requires taking inputData as a tableIndex
3613 and locating the appropriate row in the table
- 3614 3. If D1 is a TPM_DELEGATE_OWNER_BLOB or TPM_DELEGATE_KEY_BLOB, validate the
3615 integrity of D1
- 3616 a. Copy D1 -> integrityDigest to H2
- 3617 b. Set D1 -> integrityDigest to all zeros
- 3618 c. Create H3 the HMAC of D1 using tpmProof as the secret
- 3619 d. Compare H2 to H3 return TPM_AUTHFAIL on mismatch
- 3620 4. Locate (D1 -> pub -> familyID) in the TPM_FAMILY_TABLE and set familyRow to indicate
3621 row, return TPM_BADINDEX if not found
- 3622 5. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]
- 3623 6. If delegated, verify that family of the delegated Owner-auth is the same as D1:
3624 (authHandle -> familyID) == (D1 -> pub -> familyID); otherwise return error
3625 TPM_DELEGATE_FAMILY
- 3626 7. If delegated, verify that the family of the delegated Owner-auth is enabled: if (authHandle
3627 -> familyID -> flags TPM_FAMFLAG_ENABLED) is FALSE, return TPM_DISABLED_CMD
- 3628 8. Set D1 -> verificationCount to FR -> verificationCount

- 3629 9. If D1 is a TPM_DELEGATE_OWNER_BLOB or TPM_DELEGATE_KEY_BLOB set the
3630 integrity of D1
- 3631 a. Set D1 -> integrityDigest to all zeros
- 3632 b. Create H1 the HMAC of D1 using tpmProof as the secret
- 3633 c. Set D1 -> integrityDigest to H1
- 3634 10.If D1 is a blob recreate the blob and return it

3635 19.7 TPM_Delegate_VerifyDelegation

3636 Start of informative comment:

3637 TPM_VerifyDelegation interprets a delegate blob and returns success or failure, depending
3638 on whether the blob is currently valid. The delegate blob is NOT loaded into the TPM.

3639 End of informative comment.

3640 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

3641 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

3642 Actions

- 3643 1. Determine the type of blob, If delegation -> tag is equal to
3644 TPM_TAG_DELEGATE_OWNER_BLOB then
 - 3645 a. Map D1 a TPM_DELEGATE_OWNER_BLOB to delegation
- 3646 2. Else if delegation -> tag = TPM_TAG_DELG_KEY_BLOB
 - 3647 a. Map D1 a TPM_DELEGATE_KEY_BLOB to delegation
- 3648 3. Else return TPM_BAD_PARAMETER
- 3649 4. Locate D1 -> familyID in the TPM_FAMILY_TABLE and set familyRow to indicate row,
3650 return TPM_BADINDEX if not found
- 3651 5. Set FR to TPM_FAMILY_TABLE.famTableRow[familyRow]
- 3652 6. If FR -> flags TPM_FAMFLAG_ENABLED is FALSE, return TPM_DISABLED_CMD
- 3653 7. Validate that D1 -> pub -> verificationCount matches FR -> verificationCount, on
3654 mismatch return TPM_FAMILYCOUNT
- 3655 8. Validate the integrity of D1
 - 3656 a. Copy D1 -> integrityDigest to H2

- 3657 b. Set D1 -> integrityDigest to all zeros
- 3658 c. Create H3 the HMAC of D1 using tpmProof as the secret
- 3659 d. Compare H2 to H3 return TPM_AUTHFAIL on mismatch
- 3660 9. Create S1 a TPM_DELEGATE_SENSITIVE area by decrypting D1 -> sensitiveArea using
- 3661 TPM_DELEGATE_KEY
- 3662 10. Validate S1 values
- 3663 a. S1 -> tag is TPM_TAG_DELEGATE_SENSITIVE
- 3664 b. Return TPM_BAD_PARAMETER on error
- 3665 11. Return TPM_SUCCESS

3666 **20. Non-volatile Storage**

3667 **Start of informative comment:**

3668 This section handles the allocation and use of the TPM non-volatile storage.

3669 **End of informative comment.**

3670 If nvIndex refers to the DIR, the TPM ignores actions containing access control checks that
3671 have no meaning for the DIR. The TPM only checks the owner authorization.

3672

3673 **20.1 TPM_NV_DefineSpace**3674 **Start of informative comment:**

3675 This establishes the space necessary for the indicated index. The definition will include the
3676 access requirements for writing and reading the area.

3677 The space definition size does not include the area needed to manage the space.

3678 Setting TPM_PERMANENT_FLAGS -> nvLocked TRUE when it is already TRUE is not an
3679 error.

3680 **End of informative comment.**3681 **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

3682 **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

3683 **Actions**

3684 1. If pubInfo -> nvIndex == TPM_NV_INDEX_LOCK and tag = TPM_TAG_RQU_COMMAND

- 3685 a. If pubInfo -> dataSize is not 0, the command MAY return TPM_BADINDEX.
- 3686 b. Set TPM_PERMANENT_FLAGS -> nvLocked to TRUE
- 3687 c. Return TPM_SUCCESS
- 3688 2. If TPM_PERMANENT_FLAGS -> nvLocked is FALSE then all authorization checks except
- 3689 for the Max NV writes are ignored
- 3690 a. Ignored checks include physical presence, authorization, 'D' bit check, bGlobalLock,
- 3691 no authorization with a TPM owner present, bWriteSTClear, and the check that pubInfo
- 3692 -> dataSize is 0 in Action 5.c. (the no-authorization case).
- 3693 i. The check that pubInfo -> dataSize is 0 is still enforced in Action 6.f. (returning
- 3694 after deleting a previously defined storage area) and Action 9.f. (not allowing a
- 3695 space of size 0 to be defined).
- 3696 b. The check for pubInfo -> nvIndex == TPM_NV_INDEX0 in Action 3. is not ignored.
- 3697 3. If pubInfo -> nvIndex has the D bit (bit 28) set to a 1 or pubInfo -> nvIndex ==
- 3698 TPM_NV_INDEX0 then
- 3699 a. Return TPM_BADINDEX
- 3700 b. The D bit specifies an index value that is set in manufacturing and can never be
- 3701 deleted or added to the TPM
- 3702 c. Index value TPM_NV_INDEX0 is reserved and cannot be defined
- 3703 4. If tag = TPM_TAG_RQU_AUTH1_COMMAND then
- 3704 a. The TPM MUST validate the command and parameters using the TPM Owner
- 3705 authentication and ownerAuth, on error return TPM_AUTHFAIL
- 3706 b. authHandle session type MUST be OSAP
- 3707 c. Create A1 by decrypting encAuth according to the ADIP indicated by authHandle.
- 3708 5. else
- 3709 a. Validate the assertion of physical presence. Return TPM_BAD_PRESENCE on error.
- 3710 b. If TPM Owner is present then return TPM_OWNER_SET.
- 3711 c. If pubInfo -> dataSize is 0 then return TPM_BAD_DATASIZE. Setting the size to 0
- 3712 represents an attempt to delete the value without TPM Owner authentication.
- 3713 d. Validate max NV writes without an owner
- 3714 i. Set NV1 to TPM_PERMANENT_DATA -> noOwnerNVWrite
- 3715 ii. Increment NV1 by 1
- 3716 iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITES
- 3717 iv. Set TPM_PERMANENT_DATA -> noOwnerNVWrite to NV1
- 3718 e. Set A1 to encAuth. There is no nonce or authorization to create the encryption string,
- 3719 hence the AuthData value is passed in the clear
- 3720 6. If pubInfo -> nvIndex points to a valid previously defined storage area then
- 3721 a. Map D1 a TPM_NV_DATA_SENSITIVE to the storage area

- 3722 b. If D1 -> attributes specifies TPM_NV_PER_GLOBALLOCK then
- 3723 i. If TPM_STCLEAR_FLAGS -> bGlobalLock is TRUE then return
- 3724 TPM_AREA_LOCKED
- 3725 c. If D1 -> attributes specifies TPM_NV_PER_WRITE_STCLEAR
- 3726 i. If D1 -> pubInfo -> bWriteSTClear is TRUE then return TPM_AREA_LOCKED
- 3727 d. Invalidate the data area currently pointed to by D1 and ensure that if the area is
- 3728 reallocated no residual information is left
- 3729 e. The TPM invalidates authorization sessions
- 3730 i. MUST invalidate all authorization sessions associated with D1
- 3731 ii. MAY invalidate any other authorization session
- 3732 f. If pubInfo -> dataSize is 0 then return TPM_SUCCESS
- 3733 7. Parse pubInfo -> pcrInfoRead
- 3734 a. Validate pcrInfoRead structure on error return TPM_INVALID_STRUCTURE
- 3735 i. Validation includes proper PCR selections and locality selections
- 3736 8. Parse pubInfo -> pcrInfoWrite
- 3737 a. Validate pcrInfoWrite structure on error return TPM_INVALID_STRUCTURE
- 3738 i. Validation includes proper PCR selections and locality selections
- 3739 b. If pcrInfoWrite -> localityAtRelease disallows some localities
- 3740 i. Set writeLocalities to TRUE
- 3741 c. Else
- 3742 i. Set writeLocalities to FALSE
- 3743 9. Validate that the attributes are consistent
- 3744 a. The TPM SHALL ignore the bReadSTClear, bWriteSTClear and bWriteDefine
- 3745 attributes during the execution of this command
- 3746 b. If TPM_NV_PER_OWNERWRITE is TRUE and TPM_NV_PER_AUTHWRITE is TRUE
- 3747 return TPM_AUTH_CONFLICT
- 3748 c. If TPM_NV_PER_OWNERREAD is TRUE and TPM_NV_PER_AUTHREAD is TRUE
- 3749 return TPM_AUTH_CONFLICT
- 3750 d. If TPM_NV_PER_OWNERWRITE and TPM_NV_PER_AUTHWRITE and
- 3751 TPM_NV_PER_WRITEDEFINE and TPM_NV_PER_PPWRITE and writeLocalities are all
- 3752 FALSE
- 3753 i. Return TPM_PER_NOWRITE
- 3754 e. Validate pubInfo -> nvIndex
- 3755 i. Make sure that the index is applicable for this TPM. Return TPM_BADINDEX on
- 3756 error. A valid index is platform and context sensitive. That is, attempting to
- 3757 validate an index may be successful in one configuration and invalid in another
- 3758 configuration. The individual index values MUST indicate if there are any
- 3759 restrictions on the use of the index.

- 3760 ii. TPM_NV_INDEX_DIR is always an invalid defined index.
- 3761 f. If dataSize is 0 return TPM_BAD_PARAM_SIZE
- 3762 10. Create D1 a TPM_NV_DATA_SENSITIVE structure
- 3763 a. Set D1 -> pubInfo to pubInfo
- 3764 b. Set D1 -> authValue to A1
- 3765 c. Set D1 -> pubInfo -> bReadSTClear to FALSE
- 3766 d. Set D1 -> pubInfo -> bWriteSTClear to FALSE
- 3767 e. Set D1 -> pubInfo -> bWriteDefine to FALSE
- 3768 11. Validate that sufficient NV is available to store D1 and pubInfo -> dataSize bytes of data
- 3769 a. Return TPM_NOSPACE if pubInfo -> dataSize is not available in the TPM
- 3770 12. If pubInfo -> nvIndex is not TPM_NV_INDEX_TRIAL
- 3771 a. Reserve NV space for pubInfo -> dataSize
- 3772 b. Set all bytes in the newly defined area to 0xFF
- 3773 13. Ignore continueAuthSession on input and set to FALSE on output
- 3774 14. Return TPM_SUCCESS

3775 **20.2 TPM_NV_WriteValue**3776 **Start of informative comment:**

3777 This command writes the value to a defined area. The write can be TPM Owner authorized
3778 or unauthorized and protected by other attributes and will work when no TPM Owner is
3779 present.

3780 The action setting bGlobalLock to TRUE is intentionally before the action checking the
3781 owner authorization. This allows code (e.g., a BIOS) to lock NVRAM without knowing the
3782 owner authorization.

3783 **End of informative comment.**3784 **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

3785 **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

3786 **Actions**

- 3787 1. If TPM_PERMANENT_FLAGS -> nvLocked is FALSE then all authorization checks except
3788 for the max NV writes are ignored
- 3789 a. Ignored checks include physical presence, authorization,
3790 TPM_NV_PER_OWNERWRITE, PCR, bWriteDefine, bGlobalLock, bWriteSTClear, and
3791 locality.
- 3792 b. TPM_NV_PER_AUTHWRITE is not ignored.
- 3793 2. If nvIndex is TPM_NV_INDEX0 then
- 3794 a. If dataSize is not 0, the TPM MAY return TPM_BADINDEX.
- 3795 b. Set TPM_STCLEAR_FLAGS -> bGlobalLock to TRUE
- 3796 c. Return TPM_SUCCESS
- 3797 3. Locate and set D1 to the TPM_NV_DATA_AREA that corresponds to nvIndex, return
3798 TPM_BADINDEX on error
- 3799 a. If nvIndex = TPM_NV_INDEX_DIR, set D1 to TPM_PERMANENT_DATA -> authDir[0]
- 3800 4. If D1 -> permission -> TPM_NV_PER_AUTHWRITE is TRUE return
3801 TPM_AUTH_CONFLICT
- 3802 5. If tag = TPM_TAG_RQU_AUTH1_COMMAND then
- 3803 a. If D1 -> permission -> TPM_NV_PER_OWNERWRITE is FALSE return
3804 TPM_AUTH_CONFLICT
- 3805 b. Validate command and parameters using ownerAuth HMAC with TPM Owner
3806 authentication as the secret, return TPM_AUTHFAIL on error
- 3807 6. Else
- 3808 a. If D1 -> permission -> TPM_NV_PER_OWNERWRITE is TRUE return
3809 TPM_AUTH_CONFLICT
- 3810 b. If no TPM Owner validate max NV writes without an owner
- 3811 i. Set NV1 to TPM_PERMANENT_DATA -> noOwnerNVWrite
- 3812 ii. Increment NV1 by 1
- 3813 iii. If NV1 > TPM_MAX_NV_WRITE_NOOWNER return TPM_MAXNVWRITES
- 3814 iv. Set TPM_PERMANENT_DATA -> noOwnerNVWrite to NV1
- 3815 7. Check that D1 -> pcrInfoWrite -> localityAtRelease for TPM_STANY_DATA ->
3816 localityModifier is TRUE
- 3817 a. For example if TPM_STANY_DATA -> localityModifier was 2 then D1 -> pcrInfo ->
3818 localityAtRelease -> TPM_LOC_TWO would have to be TRUE
- 3819 b. On error return TPM_BAD_LOCALITY
- 3820 8. If D1 -> attributes specifies TPM_NV_PER_PPWRITE then validate physical presence is
3821 asserted if not return TPM_BAD_PRESENCE
- 3822 9. If D1 -> attributes specifies TPM_NV_PER_WRITEDEFINE
- 3823 a. If D1 -> bWriteDefine is TRUE return TPM_AREA_LOCKED
- 3824 10. If D1 -> attributes specifies TPM_NV_PER_GLOBALLOCK

- 3825 a. If TPM_STCLEAR_DATA -> bGlobalLock is TRUE return TPM_AREA_LOCKED
- 3826 11.If D1 -> attributes specifies TPM_NV_PER_WRITE_STCLEAR
- 3827 a. If D1 ->bWriteSTClear is TRUE return TPM_AREA_LOCKED
- 3828 12.If D1 -> pcrInfoWrite -> pcrSelection specifies a selection of TPM_STCLEAR_DATA ->
- 3829 PCR[]
- 3830 a. Create P1 a composite hash of the TPM_STCLEAR_DATA -> PCR[] specified by D1 ->
- 3831 pcrInfoWrite
- 3832 b. Compare P1 to D1 -> pcrInfoWrite -> digestAtRelease return TPM_WRONGPCRVAL
- 3833 on mismatch
- 3834 13.If dataSize = 0 then
- 3835 a. Set D1 -> bWriteSTClear to TRUE
- 3836 b. Set D1 -> bWriteDefine to TRUE
- 3837 14.Else
- 3838 a. Set S1 to offset + dataSize
- 3839 b. If S1 > D1 -> dataSize return TPM_NOSPACE
- 3840 c. If D1 -> attributes specifies TPM_NV_PER_WRITEALL
- 3841 i. If dataSize != D1 -> dataSize return TPM_NOT_FULLWRITE
- 3842 d. Write the new value into the NV storage area
- 3843 15.Set D1 -> bReadSTClear to FALSE
- 3844 16.Return TPM_SUCCESS

3845 **20.3 TPM_NV_WriteValueAuth**

3846 **Start of informative comment:**

3847 This command writes to a previously defined area. The area must require authorization to
3848 write. Use this command when authorization other than the owner authorization is to be
3849 used. Otherwise, use TPM_NV_WriteValue.

3850 **End of informative comment.**

3851 **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

3852 **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

3853 **Actions**

- 3854 1. Locate and set D1 to the TPM_NV_DATA_AREA that corresponds to nvIndex, return
3855 TPM_BADINDEX on error
- 3856 2. If D1 -> attributes does not specify TPM_NV_PER_AUTHWRITE then return
3857 TPM_AUTH_CONFLICT

- 3858 3. Validate authValue using D1 -> authValue, return TPM_AUTHFAIL on error
- 3859 4. Check that D1 -> pcrInfoWrite -> localityAtRelease for TPM_STANY_DATA ->
3860 localityModifier is TRUE
- 3861 a. For example if TPM_STANY_DATA -> localityModifier was 2 then D1 -> pcrInfo ->
3862 localityAtRelease -> TPM_LOC_TWO would have to be TRUE
- 3863 b. On error return TPM_BAD_LOCALITY
- 3864 5. If D1 -> attributes specifies TPM_NV_PER_PPWRITE then validate physical presence is
3865 asserted if not return TPM_BAD_PRESENCE
- 3866 6. If D1 -> pcrInfoWrite -> pcrSelection specifies a selection of PCR
- 3867 a. Create P1 a composite hash of the TPM_STCLEAR_DATA -> PCR[] specified by D1 ->
3868 pcrInfoWrite
- 3869 b. Compare P1 to digestAtRelease return TPM_WRONGPCRVAL on mismatch
- 3870 7. If D1 -> attributes specifies TPM_NV_PER_WRITEDEFINE
- 3871 a. If D1 -> bWriteDefine is TRUE return TPM_AREA_LOCKED
- 3872 8. If D1 -> attributes specifies TPM_NV_PER_GLOBALLOCK
- 3873 a. If TPM_STCLEAR_FLAGS -> bGlobalLock is TRUE return TPM_AREA_LOCKED
- 3874 9. If D1 -> attributes specifies TPM_NV_PER_WRITE_STCLEAR
- 3875 a. If D1 -> bWriteSTClear is TRUE return TPM_AREA_LOCKED
- 3876 10.If dataSize = 0 then
- 3877 a. Set D1 -> bWriteSTClear to TRUE
- 3878 b. Set D1 -> bWriteDefine to TRUE
- 3879 11.Else
- 3880 a. Set S1 to offset + dataSize
- 3881 b. If S1 > D1 -> dataSize return TPM_NOSPACE
- 3882 c. If D1 -> attributes specifies TPM_NV_PER_WRITEALL
- 3883 i. If dataSize != D1 -> dataSize return TPM_NOT_FULLWRITE
- 3884 d. Write the new value into the NV storage area
- 3885 12.Set D1 -> bReadSTClear to FALSE
- 3886 13.Return TPM_SUCCESS

3887 **20.4 TPM_NV_ReadValue**

3888 **Start of informative comment:**

3889 Read a value from the NV store. This command uses optional owner authentication.

3890 Action 1 indicates that if the NV area is not locked then reading of the NV area continues
3891 without ANY authorization. This is intentional, and allows a platform manufacturer to set
3892 the NV areas, read them back, and then lock them all without having to install a TPM
3893 owner.

3894 **End of informative comment.**

3895 **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

3896 **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

3897 **Actions**

- 3898 1. If TPM_PERMANENT_FLAGS -> nvLocked is FALSE then all authorization checks are
3899 ignored.
- 3900 a. Ignored checks include physical presence, authorization, PCR, bReadSTClear,
3901 locality, and TPM_NV_PER_OWNERREAD.
- 3902 b. TPM_NV_PER_AUTHREAD is not ignored.
- 3903 2. Set D1 a TPM_NV_DATA_AREA structure to the area pointed to by nvIndex, if not found
3904 return TPM_BADINDEX
- 3905 a. If nvIndex = TPM_NV_INDEX_DIR, set D1 to TPM_PERMANENT_DATA -> authDir[0]
- 3906 3. If tag = TPM_TAG_RQU_AUTH1_COMMAND then
- 3907 a. If D1 -> TPM_NV_PER_OWNERREAD is FALSE return TPM_AUTH_CONFLICT
- 3908 b. Validate command and parameters using TPM Owners authentication on error return
3909 TPM_AUTHFAIL
- 3910 4. Else
- 3911 a. If D1 -> TPM_NV_PER_AUTHREAD is TRUE return TPM_AUTH_CONFLICT
- 3912 b. If D1 -> TPM_NV_PER_OWNERREAD is TRUE return TPM_AUTH_CONFLICT
- 3913 5. Check that D1 -> pcrInfoRead -> localityAtRelease for TPM_STANY_DATA ->
3914 localityModifier is TRUE
- 3915 a. For example if TPM_STANY_DATA -> localityModifier was 2 then D1 -> pcrInfo ->
3916 localityAtRelease -> TPM_LOC_TWO would have to be TRUE
- 3917 b. On error return TPM_BAD_LOCALITY
- 3918 6. If D1 -> attributes specifies TPM_NV_PER_PPREAD then validate physical presence is
3919 asserted if not return TPM_BAD_PRESENCE
- 3920 7. If D1 -> TPM_NV_PER_READ_STCLEAR then
- 3921 a. If D1 -> bReadSTClear is TRUE return TPM_DISABLED_CMD
- 3922 8. If D1 -> pcrInfoRead -> pcrSelection specifies a selection of PCR
- 3923 a. Create P1 a composite hash of the TPM_STCLEAR_DATA -> PCR[] specified by D1 ->
3924 pcrInfoRead
- 3925 b. Compare P1 to D1 -> pcrInfoRead -> digestAtRelease return TPM_WRONGPCRVAL on
3926 mismatch
- 3927 9. If dataSize is 0 then
- 3928 a. Set D1 -> bReadSTClear to TRUE
- 3929 b. Set data to all zeros
- 3930 10. Else
- 3931 a. Set S1 to offset + dataSize
- 3932 b. If S1 > D1 -> dataSize return TPM_NOSPACE
- 3933 c. Set data to area pointed to by offset
- 3934 11. Return TPM_SUCCESS

3935 **20.5 TPM_NV_ReadValueAuth**

3936 **Start of informative comment:**

3937 This command requires that the read be authorized by a value set with the blob.

3938 **End of informative comment.**

3939 **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	UINT32	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

3940 **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

3941 **Actions**

- 3942 1. Locate and set D1 to the TPM_NV_DATA_AREA that corresponds to nvIndex, on error
3943 return TPM_BADINDEX
- 3944 2. If D1 -> TPM_NV_PER_AUTHREAD is FALSE return TPM_AUTH_CONFLICT
- 3945 3. Validate authHmac using D1 -> authValue on error return TPM_AUTHFAIL

- 3946 4. If D1 -> attributes specifies TPM_NV_PER_PPREAD then validate physical presence is
3947 asserted if not return TPM_BAD_PRESENCE
- 3948 5. Check that D1 -> pcrInfoRead -> localityAtRelease for TPM_STANY_DATA ->
3949 localityModifier is TRUE
- 3950 a. For example if TPM_STANY_DATA -> localityModifier was 2 then D1 -> pcrInfo ->
3951 localityAtRelease -> TPM_LOC_TWO would have to be TRUE
- 3952 b. On error return TPM_BAD_LOCALITY
- 3953 6. If D1 -> pcrInfoRead -> pcrSelection specifies a selection of PCR
- 3954 a. Create P1 a composite hash of the TPM_STCLEAR_DATA -> PCR[] specified by D1 ->
3955 pcrInfoRead
- 3956 b. Compare P1 to D1 -> pcrInfoRead -> digestAtRelease return TPM_WRONGPCRVAL on
3957 mismatch
- 3958 7. If D1 specifies TPM_NV_PER_READ_STCLEAR then
- 3959 a. If D1 -> bReadSTClear is TRUE return TPM_DISABLED_CMD
- 3960 8. If dataSize is 0 then
- 3961 a. Set D1 -> bReadSTClear to TRUE
- 3962 b. Set data to all zeros
- 3963 9. Else
- 3964 a. Set S1 to offset + dataSize
- 3965 b. If S1 > D1 -> dataSize return TPM_NOSPACE
- 3966 c. Set data to area pointed to by offset
- 3967 10. Return TPM_SUCCESS

3968 **21. Session Management**

3969 **Start of informative comment:**

3970 Three TPM_RT_CONTEXT session resources located in TPM_STANY_DATA work together to
3971 control session save and load: contextNonceSession, contextCount, and contextList[].

3972 All three MUST initialized at TPM_Startup(ST_CLEAR) and TPM_Startup(ST_DEACTIVATED)
3973 and MAY be initialized at TPM_Startup(ST_STATE). Initializing invalidates all saved
3974 sessions. They MAY be restored by TPM_Startup(ST_STATE). This case would allow saved
3975 sessions to be loaded. The actual ST_STATE operation is reported by the
3976 TPM_RT_CONTEXT startup effect.

3977 TPM_SaveContext creates a contextBlob containing an encrypted contextNonceSession. The
3978 nonce is checked by TPM_LoadContext. So initializing contextNonceSession invalidates all
3979 saved contexts. The nonce is large and protected, making a replay infeasible.

3980 The contextBlob also contains a public but protected contextCount. The count increments
3981 for each saved contextBlob. The TPM also saves contextCount in contextList[]. The TPM
3982 validates contextBlob against the contextList[] during TPM_LoadContext. Since the
3983 contextList[] is finite, it limits the number of valid saved sessions. Since the contextCount
3984 cannot be allowed to wrap, it limits the total number of saved sessions.

3985 After a contextBlob is loaded, its contextCount entry is removed from contextList[]. This
3986 releases space in the context list for future entries. It also invalidates the contextBlob. So a
3987 saved contextBlob can be loaded only once.

3988 TPM_FlushSpecific can also specify a contextCount to be removed from the contextList[],
3989 allowing invalidation of an individual contextBlob. This is different from TPM_FlushSpecific
3990 specifying a session handle, which invalidates a loaded session, not a saved contextBlob.

3991 **End of informative comment.**

3992

3993 **21.1 TPM_KeyControlOwner**

3994 **Start of informative comment:**

3995 This command controls some attributes of keys that are stored within the TPM key cache.

3996 OwnerEvict: If this bit is set to true, this key remains in the TPM non-volatile storage
3997 through all TPM_Startup events. The only way to evict this key is for the TPM Owner to
3998 execute this command again, setting the owner control bit to false and then executing
3999 TPM_FlushSpecific.

4000 The key handle does not reference an authorized entity and is not validated.

4001 **End of informative comment.**

4002 **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

4003 **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

4004 **Descriptions**

4005 1. Set an internal bit within the key cache that controls some attribute of a loaded key.

4006 **Actions**4007 1. Validate the AuthData using the owner authentication value, on error return
4008 TPM_AUTHFAIL4009 2. Validate that keyHandle refers to a loaded key, return TPM_INVALID_KEYHANDLE on
4010 error.4011 3. Validate that pubKey matches the key held by the TPM pointed to by keyHandle, return
4012 TPM_BAD_PARAMETER on mismatch4013 a. This check added so that virtualization of the keyHandle does not result in attacks as
4014 the keyHandle is not associated with an authorization value

4015 4. Validate that bitName is valid, return TPM_BAD_MODE on error.

4016 5. If bitName == TPM_KEY_CONTROL_OWNER_EVICT

4017 a. If bitValue == TRUE

- 4018 i. Verify that after this operation at least two key slots will be present within the
4019 TPM that can store any type of key both of which do NOT have the OwnerEvict bit
4020 set, on error return TPM_NOSPACE
- 4021 ii. Verify that for this key handle, parentPCRStatus is FALSE and isVolatile is
4022 FALSE. Return TPM_BAD_PARAMETER on error.
- 4023 iii. Set ownerEvict within the internal key storage structure to TRUE.
- 4024 b. Else if bitValue == FALSE
- 4025 i. Set ownerEvict within the internal key storage structure to FALSE.
- 4026 6. Return TPM_SUCCESS

4027 **21.2 TPM_SaveContext**4028 **Start of informative comment:**

4029 TPM_SaveContext saves a loaded resource outside the TPM. After successful execution of
4030 the command, the TPM automatically releases the internal memory for sessions but leaves
4031 keys in place.

4032 There is no assumption that a saved context blob is stored in a safe, protected area. Since
4033 the context blob can be loaded at any time, do not rely on TPM_SaveContext to restrict
4034 access to an entity such as a key. If use of the entity should be restricted, means such as
4035 authorization secrets or PCR's should be used.

4036 In general, TPM_SaveContext can save a transport session. However, it cannot save an
4037 exclusive transport session, because any ordinal other than TPM_ExecuteTransport
4038 terminates the exclusive transport session. This action prevents the exclusive transport
4039 session from being saved and reloaded while intervening commands are hidden from the
4040 transport log.

4041 **End of informative comment.**4042 **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

4043 **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

4044 **Description**

4045 1. The caller of the function uses the label field to add additional sequencing, anti-replay or
4046 other items to the blob. The information does not need to be confidential but needs to be
4047 part of the blob integrity.

4048 **Actions**

- 4049 1. Map V1 to TPM_STANY_DATA
- 4050 2. Validate that handle points to resource that matches resourceType, return
4051 TPM_INVALID_RESOURCE on error
- 4052 3. Validate that resourceType is a resource from the following list if not return
4053 TPM_INVALID_RESOURCE
- 4054 a. TPM_RT_KEY
- 4055 b. TPM_RT_AUTH
- 4056 c. TPM_RT_TRANS
- 4057 d. TPM_RT_DAA_TPM
- 4058 4. Locate the correct nonce
- 4059 a. If resourceType is TPM_RT_KEY
- 4060 i. If TPM_STCLEAR_DATA -> contextNonceKey is all zeros
- 4061 (1) Set TPM_STCLEAR_DATA -> contextNonceKey to the next value from the TPM
4062 RNG
- 4063 ii. Map N1 to TPM_STCLEAR_DATA -> contextNonceKey
- 4064 iii. If the key has TPM_KEY_CONTROL_OWNER_EVICT set then return
4065 TPM_OWNER_CONTROL
- 4066 b. Else
- 4067 i. If V1 -> contextNonceSession is all zeros
- 4068 (1) Set V1 -> contextNonceSession to the next value from the TPM RNG
- 4069 ii. Map N1 to V1 -> contextNonceSession
- 4070 5. Set K1 to TPM_PERMANENT_DATA -> contextKey
- 4071 6. Create R1 by putting the sensitive part of the resource pointed to by handle into a
4072 structure. The structure is a TPM manufacturer option. The TPM MUST ensure that ALL
4073 sensitive information of the resource is included in R1.
- 4074 7. Create C1 a TPM_CONTEXT_SENSITIVE structure
- 4075 a. C1 forms the inner encrypted wrapper for the blob. All saved context blobs MUST
4076 include a TPM_CONTEXT_SENSITIVE structure and the TPM_CONTEXT_SENSITIVE
4077 structure MUST be encrypted.
- 4078 b. Set C1 -> contextNonce to N1
- 4079 c. Set C1 -> internalData to R1
- 4080 8. Create B1 a TPM_CONTEXT_BLOB
- 4081 a. Set B1 -> tag to TPM_TAG_CONTEXTBLOB
- 4082 b. Set B1 -> resourceType to resourceType
- 4083 c. Set B1 -> handle to handle
- 4084 d. Set B1 -> integrityDigest to all zeros

- 4085 e. Set B1 -> label to label
- 4086 f. Set B1 -> additionalData to information determined by the TPM manufacturer. This
4087 data will help the TPM to reload and reset context. This area MUST NOT hold any data
4088 that is sensitive (symmetric IV are fine, prime factors of an RSA key are not).
- 4089 i. For OSAP sessions, and DSAP attached to keys, the hash of the entity MUST be
4090 included in additionalData
- 4091 g. Set B1 -> additionalSize to the size of additionalData
- 4092 h. Set B1 -> sensitiveSize to the size of C1
- 4093 i. Set B1 -> sensitiveData to C1
- 4094 9. If resourceType is TPM_RT_KEY
- 4095 a. Set B1 -> contextCount to 0
- 4096 10. Else
- 4097 a. If V1 -> contextCount > $2^{32}-2$ then
- 4098 i. Return with TPM_TOOMANYCONTEXTS
- 4099 b. Else
- 4100 i. Validate that the TPM can still manage the new count value
- 4101 (1) If the distance between the oldest saved context and the contextCount is too
4102 large return TPM_CONTEXT_GAP
- 4103 ii. Find contextIndex such that V1 -> contextList[contextIndex] equals 0. If not found
4104 exit with TPM_NOCONTEXTSPACE
- 4105 iii. Increment V1 -> contextCount by 1
- 4106 iv. Set V1-> contextList[contextIndex] to V1 -> contextCount
- 4107 v. Set B1 -> contextCount to V1 -> contextCount
- 4108 c. The TPM MUST invalidate all information regarding the resource except for
4109 information needed for reloading
- 4110 11. Calculate B1 -> integrityDigest the HMAC of B1 using TPM_PERMANENT_DATA ->
4111 tpmProof as the secret
- 4112 12. Create E1 by encrypting C1 using K1 as the key
- 4113 a. Set B1 -> sensitiveSize to the size of E1
- 4114 b. Set B1 -> sensitiveData to E1
- 4115 13. Set contextSize to the size of B1
- 4116 14. Return B1 in contextBlob

4117 **21.3 TPM_LoadContext**

4118 **Start of informative comment:**

4119 TPM_LoadContext loads into the TPM a previously saved context. The command returns a
4120 handle.

4121 **End of informative comment.**

4122 **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

4123 **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.

4124 **Actions**

- 4125 1. Map contextBlob to B1, a TPM_CONTEXT_BLOB structure
- 4126 2. Map V1 to TPM_STANY_DATA
- 4127 3. Create M1 by decrypting B1 -> sensitiveData using TPM_PERMANENT_DATA ->
4128 contextKey
- 4129 4. Create C1 and R1 by splitting M1 into a TPM_CONTEXT_SENSITIVE structure and
4130 internal resource data
- 4131 5. Check contextNonce
- 4132 a. If B1 -> resourceType is NOT TPM_RT_KEY
- 4133 i. If C1 -> contextNonce does not equal V1 -> contextNonceSession return
4134 TPM_BADCONTEXT

- 4135 ii. Validate that the resource pointed to by the context is loaded (i.e. for OSAP the
4136 key referenced is loaded and DSAP connected to the key) return
4137 TPM_RESOURCEMISSING
- 4138 (1) For OSAP sessions the TPM MUST validate that the incoming pubkey hash
4139 matches the key held by the TPM
- 4140 (2) For OSAP and DSAP sessions referring to a key, verify that entityHandle
4141 identifies the key linked to this OSAP/DSAP session, if not return
4142 TPM_BAD_HANDLE.
- 4143 b. Else
- 4144 i. If C1 -> internalData -> parentPCRStatus is FALSE and C1 -> internalData ->
4145 isVolatile is FALSE
- 4146 (1) Ignore C1 -> contextNonce
- 4147 ii. else
- 4148 (1) If C1 -> contextNonce does not equal TPM_STCLEAR_DATA ->
4149 contextNonceKey return TPM_BADCONTEXT
- 4150 6. Validate the structure
- 4151 a. Set H1 to B1 -> integrityDigest
- 4152 b. Set B1 -> integrityDigest to all zeros
- 4153 c. Copy M1 to B1 -> sensitiveData
- 4154 d. Create H2 the HMAC of B1 using TPM_PERMANENT_DATA -> tpmProof as the HMAC
4155 key
- 4156 e. If H2 does not equal H1 return TPM_BADCONTEXT
- 4157 7. If keepHandle is TRUE
- 4158 a. Set handle to B1 -> handle
- 4159 b. If the TPM is unable to restore the handle the TPM MUST return TPM_BAD_HANDLE
- 4160 8. Else
- 4161 a. The TPM SHOULD attempt to restore the handle but if not possible it MAY set the
4162 handle to any valid for B1 -> resourceType
- 4163 9. If B1 -> resourceType is NOT TPM_RT_KEY
- 4164 a. Find contextIndex such that V1 -> contextList[contextIndex] equals B1 ->
4165 TPM_CONTEXT_BLOB -> contextCount
- 4166 b. If not found then return TPM_BADCONTEXT
- 4167 c. Set V1 -> contextList[contextIndex] to 0
- 4168 10. Process B1 to return the resource back into TPM use

4169 **22. Eviction**

4170 **Start of informative comment:**

4171 The TPM has numerous resources held inside of the TPM that may need eviction. The need
4172 for eviction occurs when the number of resources in use by the TPM exceed the available
4173 space. For resources that are hard to reload (i.e. keys tied to PCR values) the outside entity
4174 should first perform a context save before evicting items.

4175 In version 1.1 there were separate commands to evict separate resource types. This new
4176 command set uses the resource types defined for context saving and creates a generic
4177 command that will evict all resource types.

4178 **End of informative comment.**

4179 The TPM MUST NOT flush the EK or SRK using this command.

4180 Version 1.2 deprecates the following commands:

- 4181 • TPM_Terminate_Handle
- 4182 • TPM_EvictKey
- 4183 • TPM_Reset

4184 **22.1 TPM_FlushSpecific**4185 **Start of informative comment:**

4186 TPM_FlushSpecific flushes from the TPM a specific handle.

4187 **End of informative comment.**4188 **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

4189 **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

4190 **Description**

4191 TPM_FlushSpecific releases the resources associated with the given handle.

4192 **Actions**

4193 1. If resourceType is TPM_RT_CONTEXT

4194 a. The handle for a context is not a handle but the "context count" value. The TPM uses
4195 the "context count" value to locate the proper contextList entry and sets R1 to the
4196 contextList entry

4197 2. Else if resourceType is TPM_RT_KEY

4198 a. Set R1 to the key pointed to by handle

4199 b. If R1 -> ownerEvict is TRUE return TPM_KEY_OWNER_CONTROL

4200 3. Else if resourceType is TPM_RT_AUTH

4201 a. Set R1 to the authorization session pointed to by handle

4202 4. Else if resourceType is TPM_RT_TRANS

4203 a. Set R1 to the transport session pointed to by handle

4204 5. Else if resourceType is TPM_RT_DAA_TPM

- 4205 a. Set R1 to the DAA session pointed to by handle
- 4206 6. Else return TPM_INVALID_RESOURCE
- 4207 7. Validate that R1 determined by resourceType and handle points to a valid allocated
- 4208 resource. Return TPM_BAD_PARAMETER on error.
- 4209 8. Invalidate R1 and all internal resources allocated to R1
- 4210 a. Resources include authorization sessions

4211 **23. Timing Ticks**4212 **Start of informative comment:**

4213 The TPM timing ticks are always available for use. The association of timing ticks to actual
4214 time is a protocol that occurs outside of the TPM. See the design document for details.

4215 The setting of the clock type variable is a one time operation that allows the TPM to be
4216 configured to the type of platform that is installed on.

4217 The ability for the TPM to continue to increment the timer ticks across power cycles of the
4218 platform is a TPM and platform manufacturer decision.

4219 **End of informative comment.**4220 **23.1 TPM_GetTicks**4221 **Start of informative comment:**

4222 This command returns the current tick count of the TPM.

4223 **End of informative comment.**4224 **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

4225 **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

4226 **Descriptions**

4227 This command returns the current time held in the TPM. It is the responsibility of the
4228 external system to maintain any relation between this time and a UTC value or local real
4229 time value.

4230 **Actions**

- 4231 1. Set T1 to the internal TPM_CURRENT_TICKS structure
- 4232 2. Return T1 as currentTime.

4233 **23.2 TPM_TickStampBlob**

4234 **Start of informative comment:**

4235 This command applies a time stamp to the passed blob. The TPM makes no representation
4236 regarding the blob merely that the blob was present at the TPM at the time indicated.

4237 **End of informative comment.**

4238 **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

4239 **Outgoing Parameters and Sizes**

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

4240 **Description**

4241 The function performs a digital signature on the hash of digestToStamp and the current tick
4242 count.

4243 It is the responsibility of the external system to maintain any relation between tick count
4244 and a UTC value or local real time value.

4245 **Actions**

- 4246 1. The TPM validates the AuthData to use the key pointed to by keyHandle.
- 4247 2. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING, TPM_KEY_IDENTITY or
4248 TPM_KEY_LEGACY, if not return the error code TPM_INVALID_KEYUSAGE.
- 4249 3. Validate that keyHandle -> sigScheme is TPM_SS_RSASSAPKCS1v15_SHA1 or
4250 TPM_SS_RSASSAPKCS1v15_INFO, if not return TPM_INAPPROPRIATE_SIG.
- 4251 4. If TPM_STCLEAR_DATA -> currentTicks is not properly initialized
 - 4252 a. Initialize the TPM_STCLEAR_DATA -> currentTicks
- 4253 5. Create T1, a TPM_CURRENT_TICKS structure.
- 4254 6. Create H1 a TPM_SIGN_INFO structure and set the structure defaults
 - 4255 a. Set H1 -> fixed to "TSTP"
 - 4256 b. Set H1 -> replay to antiReplay
 - 4257 c. Create H2 the concatenation of digestToStamp || T1
 - 4258 d. Set H1 -> dataLen to the length of H2
 - 4259 e. Set H1 -> data to H2

- 4260 7. The TPM computes the signature, sig, using the key referenced by keyHandle, using
4261 SHA-1 of H1 as the information to be signed
- 4262 8. The TPM returns T1 as currentTicks parameter

4263 **24. Transport Sessions**4264 **24.1 TPM_EstablishTransport**4265 **Start of informative comment:**

4266 This establishes the transport session. Depending on the attributes specified for the session
4267 this may establish shared secrets, encryption keys, and session logs. The session will be in
4268 use for by the TPM_ExecuteTransport command.

4269 The only restriction on what can happen inside of a transport session is that there is no
4270 “nesting” of sessions. It is permissible to perform operations that delete internal state and
4271 make the TPM inoperable.

4272 **End of informative comment.**4273 **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

4274

4275 **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

4276 **Description**

4277 This command establishes the transport sessions shared secret. The encryption of the
4278 shared secret uses the public key of the key loaded in encKey.

4279 **Actions**

- 4280 1. If encHandle is TPM_KH_TRANSPORT then
- 4281 a. If tag is NOT TPM_TAG_RQU_COMMAND return TPM_BADTAG
- 4282 b. If transPublic -> transAttributes specifies TPM_TRANSPORT_ENCRYPT return
4283 TPM_BAD_SCHEME
- 4284 c. If secretSize is not 20 return TPM_BAD_PARAM_SIZE
- 4285 d. Set A1 to secret
- 4286 2. Else
- 4287 a. encHandle -> keyUsage MUST be TPM_KEY_STORAGE or TPM_KEY_LEGACY return
4288 TPM_INVALID_KEYUSAGE on error
- 4289 b. If encHandle -> authDataUsage does not equal TPM_AUTH_NEVER and tag is NOT
4290 TPM_TAG_RQU_AUTH1_COMMAND return TPM_AUTHFAIL
- 4291 c. Using encHandle -> usageAuth validate the AuthData to use the key and the
4292 parameters to the command
- 4293 d. Create K1 a TPM_TRANSPORT_AUTH structure by decrypting secret using the key
4294 pointed to by encHandle
- 4295 e. Validate K1 for tag
- 4296 f. Set A1 to K1 -> authData
- 4297 3. If transPublic -> transAttributes has TPM_TRANSPORT_ENCRYPT

- 4298 a. If TPM_PERMANENT_FLAGS -> FIPS is true and transPublic -> algId is equal to
4299 TPM_ALG_MGF1 return TPM_INAPPROPRIATE_ENC
- 4300 b. Check if the transPublic -> algId is supported, if not return
4301 TPM_BAD_KEY_PROPERTY
- 4302 c. If transPublic -> algId is TPM_ALG_AESXXX, check that transPublic -> encScheme is
4303 supported, if not return TPM_INAPPROPRIATE_ENC
- 4304 d. Perform any initializations necessary for the algorithm
- 4305 4. Generate transNonceEven from the TPM RNG
- 4306 5. Create T1 a TPM_TRANSPORT_INTERNAL structure
- 4307 a. Ensure that the TPM has sufficient internal space to allocate the transport session,
4308 return TPM_RESOURCES on error
- 4309 b. Assign a T1 -> transHandle value. This value is assigned by the TPM
- 4310 c. Set T1 -> transDigest to all zeros
- 4311 d. Set T1 -> transPublic to transPublic
- 4312 e. Set T1-> transNonceEven to transNonceEven
- 4313 f. Set T1 -> authData to A1
- 4314 6. If TPM_STANY_DATA -> currentTicks is not properly initialized
- 4315 a. Initialize the TPM_STANY_DATA -> currentTicks
- 4316 7. Set currentTicks to TPM_STANY_DATA -> currentTicks
- 4317 8. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_LOG set then
- 4318 a. Create L1 a TPM_TRANSPORT_LOG_IN structure
- 4319 i. Set L1 -> parameters to SHA-1 (ordinal || transPublic || secretSize || secret)
- 4320 ii. Set L1 -> pubKeyHash to all zeros
- 4321 iii. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || L1)
- 4322 b. Create L2 a TPM_TRANSPORT_LOG_OUT structure
- 4323 i. Set L2 -> parameters to SHA-1 (returnCode || ordinal || locality || currentTicks
4324 || transNonceEven)
- 4325 ii. Set L2 -> locality to the locality of this command
- 4326 iii. Set L2 -> currentTicks to currentTicks, this MUST be the same value that is
4327 returned in the currentTicks parameter
- 4328 iv. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || L2)
- 4329 9. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_EXCLUSIVE then set
4330 TPM_STANY_FLAGS -> transportExclusive to TRUE
- 4331 a. Execution of any command other than TPM_ExecuteTransport or
4332 TPM_ReleaseTransportSigned targeting this transport session will cause the abnormal
4333 invalidation of this transport session transHandle

- 4334 b. The TPM gives no indication, other than invalidation of transHandle, that the session
4335 is terminated
- 4336 10.Return T1 -> transHandle as transHandle

4337 **24.2 TPM_ExecuteTransport**4338 **Start of informative comment:**

4339 Delivers a wrapped TPM command to the TPM where the TPM unwraps the command and
4340 then executes the command.

4341 TPM_ExecuteTransport uses the same rolling nonce paradigm as other authorized TPM
4342 commands. The even nonces start in TPM_EstablishTransport and change on each
4343 invocation of TPM_ExecuteTransport.

4344 The only restriction on what can happen inside of a transport session is that there is no
4345 “nesting” of sessions. It is permissible to perform operations that delete internal state and
4346 make the TPM inoperable.

4347 Because, in general, key handles are not logged, a digest of the corresponding public key is
4348 logged. In cases where the key handle is logged (e.g. TPM_OwnerReadInternalPub), the
4349 public key is also logged.

4350 The method of incrementing the symmetric key counter value is different from that used by
4351 some standard crypto libraries (e.g. openssl, Java JCE) that increment the entire counter
4352 value. TPM users should be aware of this to avoid errors when the counter wraps.

4353 **End of informative comment.**4354 **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

4355

4356 **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

4357 **Description**

- 4358 1. This command executes a TPM command using the transport session.
- 4359 2. Prior to execution of the wrapped command (action 11 below) failure of the transport
4360 session **MUST** have no effect on the resources referenced by the wrapped command. The
4361 exception is when the TPM goes into failure mode and return FAILED_SELFTEST for all
4362 subsequent commands.
- 4363 3. After execution of the wrapped command, failure of the transport session **MAY NOT**
4364 affect wrapped command resources. That is, the TPM is not required to clean up the
4365 effects of the wrapped command. Sessions and keys **MAY** remain loaded. It is
4366 understood that the transport session will be returning an error code and not reporting
4367 any session nonces. Therefore, wrapped sessions are no longer useful to the caller. It is
4368 the responsibility of the caller to clean up the result of the wrapped command.
- 4369 4. Execution of the wrapped command (action 11) **SHOULD** have no effect on the transport
4370 session.
- 4371 a. The wrapped command **SHALL** use no resources of the transport session, this
4372 includes authorization sessions
- 4373 b. If the wrapped command execution returns an error (action 11 below) then the
4374 sessions for TPM_ExecuteTransport still operate properly.
- 4375 c. The exception to this is when the wrapped command causes the TPM to go into
4376 failure mode and return TPM_FAILSELFTEST for all subsequent commands
- 4377 5. Field layout
- 4378 a. Notation
- 4379 i. et indicates the outer TPM_ExecuteTransport command and response

4380 ii. w indicates the inner command and response that is wrapped by the
4381 TPM_ExecuteTransport.

4382 iii. (o) indicates optional parameters that may or may not be present in the wrapped
4383 command.

4384 b. Command representation

4385 c. *****

4386 d. TAGet | LENet | ORDet | wrappedCmdSize | wrappedCmd | AUTHet

4387 e. *****

4388 f. wrappedCmd looks like the following

4389 g. *****

4390 h. TAGw | LENw | ORDw | HANDLESw(o) | DATAw | AUTH1w (o) | AUTH2w (o)

4391 i. *****

4392 j. | LEN1 |

4393 k. | E1 | (encrypted)

4394 l. | C1 | (decrypted)

4395 m. Response representation

4396 n. *****

4397 o. TAGet | LENet | RCet | wrappedRspSize | wrappedRsp | AUTHet

4398 p. *****

4399 q. wrappedRsp looks like the following

4400 r. *****

4401 s. TAGw | LENw | RCw | HANDLESw(o) | DATAw | AUTH1w (o) | AUTH2w (o)

4402 t. *****

4403 u. | LEN2 |

4404 v. | ←----- C2 -----→ |

4405 w. | S2 | (decrypted)

4406 x. | E2 | (encrypted)

4407 y. The only command and response parameter that is possibly encrypted is DATAw.

4408 6. Additional DATAw comments

4409 a. For TPM_FlushSpecific and TPM_SaveContext

4410 i. The DATAw part of these commands does not include the handle.

4411 (1) It is understood that encrypting the resourceType prevents a determination of
4412 the handle type.

4413 ii. If the resourceType is TPM_RT_KEY, then the public key SHOULD be logged.

4414 b. For TPM_DAA_Join and TPM_DAA_Sign

- 4415 i. The DATAw part of these commands does not include the input handle. The
4416 output handle from stage 0 is included in DATAw.
- 4417 c. For TPM_LoadKey2
- 4418 i. The outgoing handle is not part of the outgoing DATAw and is not encrypted or
4419 logged by the outgoing transport.
- 4420 d. For TPM_LoadKey
- 4421 i. The outgoing handle is part of the outgoing DATAw and is encrypted and logged.
- 4422 e. For TPM_LoadContext
- 4423 i. The outgoing handle is not part of the outgoing DATAw and is not encrypted or
4424 logged by the outgoing transport.
- 4425 (1) It is understood that encrypting the contextBlob prevents a determination of
4426 the handle type.
- 4427 7. TPM_ExecuteTransport returns an implementation defined result when the wrapped
4428 command would cause termination of the transport session. Implementation defined
4429 possibilities include but are not limited to: the wrapped command may execute,
4430 completely, partially, or not at all, the transport session may or may not be terminated,
4431 continueTransSession may not be processed or returned correctly, and an error may or
4432 may not be returned. The wrapped commands include:
 - 4433 a. TPM_FlushSpecific, TPM_SaveContext targeting the transport session
 - 4434 b. TPM_OwnerClear, TPM_ForceClear, TPM_RevokeTrust

4435 Actions

- 4436 1. Using transHandle locate the TPM_TRANSPORT_INTERNAL structure T1
- 4437 2. Parse wrappedCmd
 - 4438 a. Set TAGw, LENw, and ORDw to the parameters from wrappedCmd
 - 4439 b. Set E1 to DATAw
 - 4440 i. This pointer is ordinal dependent and requires the execute transport command to
4441 parse wrappedCmd
 - 4442 c. Set LEN1 to the length of DATAw
 - 4443 i. DATAw always ends at the start of AUTH1w if AUTH1w is present
- 4444 3. If LEN1 is less than 0, or if ORDw is unknown, unimplemented, or cannot be determined
 - 4445 a. Return TPM_BAD_PARAMETER
- 4446 4. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_ENCRYPT set then
 - 4447 a. If T1 -> transPublic -> algId is TPM_ALG_MGF1
 - 4448 i. Using the MGF1 function, create string G1 of length LEN1. The inputs to the
4449 MGF1 are transLastNonceEven, transNonceOdd, "in", and T1 -> authData. These
4450 four values concatenated together form the Z value that is the seed for the MGF1.
 - 4451 ii. Create C1 by performing an XOR of G1 and wrappedCmd starting at E1.
 - 4452 b. If the encryption algorithm requires an IV or CTR, calculate the IV or CTR value

- 4453 i. Using the MGF1 function, create string IV1 or CTR1 with a length set by the block
4454 size of the encryption algorithm. The inputs to the MGF1 are
4455 transLastNonceEven, transNonceOdd, and “in”. These three values concatenated
4456 together form the Z value that is the seed for the MGF1. Note that any
4457 terminating characters within the string “in” are ignored, so a total of 42 bytes are
4458 hashed.
- 4459 ii. The symmetric key is taken from the first bytes of T1 -> authData.
- 4460 iii. Decrypt DATAw and replace the DATAw area of E1 creating C1
- 4461 c. TPM_OSAP, TPM_OIAP have no parameters encrypted
- 4462 d. TPM_DSAP has special rules for parameter encryption
- 4463 5. Else
- 4464 a. Set C1 to the DATAw area E1 of wrappedCmd
- 4465 6. Create H1 the SHA-1 of (ORDw || C1).
- 4466 a. C1 MUST point at the decrypted DATAw area of E1
- 4467 b. The TPM MAY use this calculation for both execute transport authorization,
4468 authorization of the wrapped command and transport log creation
- 4469 7. Validate the incoming transport session authorization
- 4470 a. Set inParamDigest to SHA-1 (ORDet || wrappedCmdSize || H1)
- 4471 b. Calculate the HMAC of (inParamDigest || transLastNonceEven || transNonceOdd ||
4472 continueTransSession) using T1 -> authData as the HMAC key
- 4473 c. Validate transAuth, on errors return TPM_AUTHFAIL
- 4474 8. If TPM_ExecuteTransport requires auditing
- 4475 a. Create TPM_AUDIT_EVENT_IN using H1 as the input parameter digest and update
4476 auditDigest
- 4477 b. On any error return TPM_AUDITFAIL_UNSUCCESSFUL
- 4478 9. If ORDw is from the list of following commands return TPM_NO_WRAP_TRANSPORT
- 4479 a. TPM_EstablishTransport
- 4480 b. TPM_ExecuteTransport
- 4481 c. TPM_ReleaseTransportSigned
- 4482 10. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_LOG set then
- 4483 a. Create L2 a TPM_TRANSPORT_LOG_IN structure
- 4484 b. Set L2 -> parameters to H1
- 4485 c. If ORDw is a command with no key handles
- 4486 i. Set L2 -> pubKeyHash to all zeros
- 4487 d. If ORDw is a command with one key handle
- 4488 i. Create K2 the hash of the TPM_STORE_PUBKEY structure of the key pointed to
4489 by the key handle.

- 4490 ii. Set L2 -> pubKeyHash to SHA-1 (K2)
- 4491 e. If ORDw is a command with two key handles
- 4492 i. Create K2 the hash of the TPM_STORE_PUBKEY structure of the key pointed to
4493 by the first key handle.
- 4494 ii. Create K3 the hash of the TPM_STORE_PUBKEY structure of the key pointed to
4495 by the second key handle.
- 4496 iii. Set L2 -> pubKeyHash to SHA-1 (K2 || K3)
- 4497 f. Set T1 -> transDigest to the SHA-1 (T1 -> transDigest || L2)
- 4498 g. If ORDw is a command with key handles, and the key is not loaded, return
4499 TPM_INVALID_KEYHANDLE.
- 4500 11. Send the wrapped command to the normal TPM command parser, the output is C2 and
4501 the return code is RCw
- 4502 a. If ORDw is a command that is audited then the TPM MUST perform the input and
4503 output audit of the command as part of this action.
- 4504 b. The TPM MAY use H1 as the data value in the authorization and audit calculations
4505 during the execution of C1
- 4506 12. Set CT1 to TPM_STANY_DATA -> currentTicks -> currentTicks and return CT1 in the
4507 currentTicks output parameter
- 4508 13. Calculate S2 the pointer to the DATAw area of C2
- 4509 a. Calculate LEN2 the length of S2 according to the same rules that calculated LEN1
- 4510 14. Create H2 the SHA-1 of (RCw || ORDw || S2)
- 4511 a. The TPM MAY use this calculation for execute transport authorization and transport
4512 log out creation
- 4513 15. Calculate the outgoing transport session authorization
- 4514 a. Create the new transNonceEven for the output of the command
- 4515 b. Set outParamDigest to SHA-1 (RCet || ORDet || TPM_STANY_DATA -> currentTicks
4516 -> currentTicks || locality || wrappedRspSize || H2)
- 4517 c. Calculate transAuth, the HMAC of (outParamDigest || transNonceEven ||
4518 transNonceOdd || continueTransSession) using T1 -> authData as the HMAC key
- 4519 16. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_LOG set then
- 4520 a. Create L3 a TPM_TRANSPORT_LOG_OUT structure
- 4521 b. Set L3 -> parameters to H2
- 4522 c. Set L3 -> currentTicks to TPM_STANY_DATA -> currentTicks
- 4523 d. Set L3 -> locality to TPM_STANY_DATA -> localityModifier
- 4524 e. Set T1 -> transDigest to the SHA-1 (T1 -> transDigest || L3)
- 4525 17. If T1 -> transPublic -> transAttributes has TPM_TRANSPORT_ENCRYPT set then
- 4526 a. If T1 -> transPublic -> algId is TPM_ALG_MGF1

- 4527 i. Using the MGF1 function, create string G2 of length LEN2. The inputs to the
4528 MGF1 are transNonceEven, transNonceOdd, “out”, and T1 -> authData. These
4529 four values concatenated together form the Z value that is the seed for the MGF1.
- 4530 ii. Create E2 by performing an XOR of G2 and C2 starting at S2.
- 4531 b. Else
- 4532 i. Create IV2 or CTR2 using the same algorithm as IV1 or CTR1 with the input
4533 values transNonceEven, transNonceOdd, and “out”. Note that any terminating
4534 characters within the string “out” are ignored, so a total of 43 bytes are hashed.
- 4535 ii. The symmetric key is taken from the first bytes of T1 -> authData
- 4536 iii. Create E2 by encrypting C2 starting at S2
- 4537 18.Else
- 4538 a. Set E2 to the DATAw area S2 of wrappedRsp
- 4539 19.If continueTransSession is FALSE
- 4540 a. Invalidate all session data related to transHandle
- 4541 20.If TPM_ExecuteTransport requires auditing
- 4542 a. Create TPM_AUDIT_EVENT_OUT using H2 for the parameters and update the
4543 auditDigest
- 4544 b. On any errors return TPM_AUDITFAIL_SUCCESSFUL or
4545 TPM_AUDITFAIL_UNSUCCESSFUL depending on RCw
- 4546 21.Return C2 but with S2 replaced by E2 in the wrappedRsp parameter

4547 **24.3 TPM_ReleaseTransportSigned**

4548 **Start of informative comment:**

4549 This command completes the transport session. If logging for this session is turned on, then
4550 this command returns a hash of all operations performed during the session along with a
4551 digital signature of the hash.

4552 This command serves no purpose if logging is turned off, and results in an error if
4553 attempted.

4554 This command uses two authorization sessions, the key that will sign the log and the
4555 authorization from the session. Having the session authorization proves that the requestor
4556 that is signing the log is the owner of the session. If this restriction is not put in then an
4557 attacker can close the log and sign using their own key.

4558 The hash of the session log includes the information associated with the input phase of
4559 execution of the TPM_ReleaseTransportSigned command. It cannot include the output
4560 phase information.

4561 **End of informative comment.**

4562 **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

4563 **Outgoing Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReleaseTransportSigned
4	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

4564 **Description**

4565 This command releases a transport session and signs the transport log

4566 **Actions**

- 4567 1. Using transHandle locate the TPM_TRANSPORT_INTERNAL structure T1
- 4568 2. Validate that keyHandle -> sigScheme is TPM_SS_RSASSAPKCS1v15_SHA1 or
4569 TPM_SS_RSASSAPKCS1v15_INFO, if not return TPM_INAPPROPRIATE_SIG.
- 4570 3. Validate that keyHandle -> keyUsage is TPM_KEY_SIGNING, if not return
4571 TPM_INVALID_KEYUSAGE
- 4572 4. Using key -> authData validate the command and parameters, on error return
4573 TPM_AUTHFAIL
- 4574 5. Using transHandle -> authData validate the command and parameters, on error return
4575 TPM_AUTH2FAIL
- 4576 6. If T1 -> transAttributes has TPM_TRANSPORT_LOG set then
- 4577 a. Create A1 a TPM_TRANSPORT_LOG_OUT structure
- 4578 b. Set A1 -> parameters to the SHA-1 (ordinal || antiReplay)
- 4579 c. Set A1 -> currentTicks to TPM_STANY_DATA -> currentTicks
- 4580 d. Set A1 -> locality to the locality modifier for this command

- 4581 e. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || A1)
- 4582 7. Else
 - 4583 a. Return TPM_BAD_MODE
- 4584 8. Create H1 a TPM_SIGN_INFO structure and set the structure defaults
 - 4585 a. Set H1 -> fixed to "TRAN"
 - 4586 b. Set H1 -> replay to antiReplay
 - 4587 c. Set H1 -> data to T1 -> transDigest
 - 4588 d. Sign SHA-1 hash of H1 using the key pointed to by keyHandle
- 4589 9. Invalidate all session data related to T1
- 4590 10. Set continueTransSession to FALSE
- 4591 11. Return TPM_SUCCESS

4592 **25. Monotonic Counter**4593 **25.1 TPM_CreateCounter**4594 **Start of informative comment:**

4595 This command creates the counter but does not select the counter. Counter creation
4596 assigns an AuthData value to the counter and sets the counters original start value. The
4597 original start value is the current internal base value plus one. Setting the new counter to
4598 the internal base avoids attacks on the system that are attempting to use old counter
4599 values.

4600 **End of informative comment.**4601 **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.

4602

4603 **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.

4604 **Description**

4605 This command creates a new monotonic counter. The TPM MUST support a minimum of 4
4606 concurrent counters.

4607 **Actions**

4608 The TPM SHALL do the following:

- 4609 1. Using the authHandle field, validate the owner's AuthData to execute the command and
4610 all of the incoming parameters. The authorization session MUST be OSAP or DSAP
- 4611 2. Ignore continueAuthSession on input and set continueAuthSession to FALSE on output
- 4612 3. Create a1 by decrypting encAuth according to the ADIP indicated by authHandle.
- 4613 4. Validate that there is sufficient internal space in the TPM to create a new counter. If
4614 there is insufficient space, the command returns an error.
 - 4615 a. The TPM MUST provide storage for a1, TPM_COUNTER_VALUE, countID, and any
4616 other internal data the TPM needs to associate with the counter
- 4617 5. Increment the max counter value
- 4618 6. Set the counter to the max counter value
- 4619 7. Set the counter label to label
- 4620 8. Create a countID

4621 **25.2 TPM_IncrementCounter**4622 **Start of informative comment:**

4623 This authorized command increments the indicated counter by one. Once a counter has
4624 been incremented then all subsequent increments must be for the same handle until a
4625 successful TPM_Startup(ST_CLEAR) is executed.

4626 The order for checking validation of the command parameters when no counter is active,
4627 keeps an attacker from creating a denial-of-service attack.

4628 **End of informative comment.**4629 **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

4630 **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

4631 **Description**

4632 This function increments the counter by 1.

4633 The TPM MAY implement increment throttling to avoid burn problems

4634 **Actions**

- 4635 1. If TPM_STCLEAR_DATA -> countID is 0
- 4636 a. Validate that countID is a valid counter, return TPM_BAD_COUNTER on mismatch
- 4637 b. Validate the command parameters using counterAuth
- 4638 c. Set TPM_STCLEAR_DATA -> countID to countID
- 4639 2. else
- 4640 a. If TPM_STCLEAR_DATA -> countID does not equal countID
- 4641 i. Return TPM_BAD_COUNTER
- 4642 b. Validate the command parameters using counterAuth
- 4643 3. Increments the counter by 1
- 4644 4. Return new count value in count

4645 **25.3 TPM_ReadCounter**4646 **Start of informative comment:**

4647 Reading the counter provides the caller with the current number in the sequence.

4648 **End of informative comment.**4649 **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

4650 **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	4	TPM_COUNTER_VALUE	count	The counter value

4651 **Description**4652 This returns the current value for the counter indicated. The counter MAY be any valid
4653 counter.4654 **Actions**

4655 1. Validate that countID points to a valid counter. Return TPM_BAD_COUNTER on error.

4656 2. Return count

4657 **25.4 TPM_ReleaseCounter**

4658 **Start of informative comment:**

4659 This command releases a counter such that no reads or increments of the indicated counter
4660 will succeed.

4661 **End of informative comment.**

4662 **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

4663 **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

4664 **Actions**

4665 The TPM uses countID to locate a valid counter.

4666 1. Authenticate the command and the parameters using the AuthData pointed to by
4667 countID. Return TPM_AUTHFAIL on error

4668 2. The TPM invalidates all internal information regarding the counter. This includes
4669 releasing countID such that any subsequent attempts to use countID will fail.

4670 3. The TPM invalidates sessions

- 4671 a. MUST invalidate all OSAP sessions associated with the counter
- 4672 b. MAY invalidate any other session
- 4673 4. If TPM_STCLEAR_DATA -> countID equals countID,
- 4674 a. Set TPM_STCLEAR_DATA -> countID to an illegal value (not the zero value)

4675 **25.5 TPM_ReleaseCounterOwner**

4676 **Start of informative comment:**

4677 This command releases a counter such that no reads or increments of the indicated counter
4678 will succeed.

4679 **End of informative comment.**

4680 **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

4681 **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

4682 **Description**

4683 This invalidates all information regarding a counter.

4684 **Actions**

- 4685 1. Validate that ownerAuth properly authorizes the command and parameters
- 4686 2. The TPM uses countID to locate a valid counter. Return TPM_BAD_COUNTER if not
4687 found.

- 4688 3. The TPM invalidates all internal information regarding the counter. This includes
4689 releasing countID such that any subsequent attempts to use countID will fail.
- 4690 4. The TPM invalidates sessions
- 4691 a. MUST invalidate all OSAP sessions associated with the counter
- 4692 b. MAY invalidate any other session
- 4693 5. If TPM_STCLEAR_DATA -> countID equals countID,
- 4694 a. Set TPM_STCLEAR_DATA -> countID to an illegal value (not the zero value)

4695 **26. DAA commands**

4696 **26.1 TPM_DAA_Join**

4697 **Start of informative comment:**

4698 TPM_DAA_Join is the process that establishes the DAA parameters in the TPM for a specific
4699 DAA issuing authority.

4700 outputSize and outputData are always included in the outParamDigest. This includes stage
4701 0, where the outputData contains the DAA session handle.

4702 **End of informative comment.**

4703 **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.

4704 **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.

4705 **Description**

4706 This table summaries the input, output and saved data that is associated with each stage of
4707 processing.

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

4708

4709 **Actions**

4710 A Trusted Platform Module that receives a valid TPM_DAA_Join command SHALL:

- 4711 1. Use ownerAuth to verify that the Owner authorized all TPM_DAA_Join input parameters.
- 4712 2. Any error return results in the TPM invalidating all resources associated with the join
- 4713 3. Constant values of 0 or 1 are 1 byte integers, stages affected are
 - 4714 a. 4(j), 5(j), 14(f), 17(e)
- 4715 4. Representation of the strings “r0” to “r4” are 2-byte ASCII encodings, stages affected are
 - 4716 a. 9(i), 10(h), 11(h), 12(h), 15(f),15(g), 17(d), 18(d), 19(d), 20(d), 21(d)

4717 **Start of informative comment:**

4718 5. Variable DAA_Count

- 4719 a. In stage 0, DAA_Count denotes the length of the RSA key chain, which certifies the
 4720 main DAA public key and which will be loaded in stage 1. It also denotes the number of
 4721 times stage 1 is executed.
- 4722 b. In stage 3 the variable DAA_count denotes the actual DAA counter. It allows a DAA
 4723 issuer to keep track of the number of times it has issued 'different' DAA credentials to
 4724 the same platform. (The counter does not need to be equal to the actual number.)

4725 **End of informative comment.**4726 **Stages**

4727 0. If stage==0

- 4728 a. Determine that sufficient resources are available to perform a TPM_DAA_Join.
 - 4729 i. The TPM MUST support sufficient resources to perform one (1) TPM_DAA_Join/
 4730 TPM_DAA_Sign. The TPM MAY support additional TPM_DAA_Join/
 4731 TPM_DAA_Sign sessions.
 - 4732 ii. The TPM may share internal resources between the DAA operations and other
 4733 variable resource requirements:
 - 4734 iii. If there are insufficient resources within the stored key pool (and one or more
 4735 keys need to be removed to permit the DAA operation to execute) return
 4736 TPM_NOSPACE
 - 4737 iv. If there are insufficient resources within the stored session pool (and one or
 4738 more authorization or transport sessions need to be removed to permit the
 4739 DAA operation to execute), return TPM_RESOURCES.
- 4740 b. Set all fields in DAA_issuerSettings = NULL
- 4741 c. set all fields in DAA_tpmSpecific = NULL
- 4742 d. set all fields in DAA_session = NULL
- 4743 e. Set all fields in DAA_joinSession = NULL
- 4744 f. Verify that sizeof(inputData0) == sizeof(DAA_tpmSpecific -> DAA_count) and return
 4745 error TPM_DAA_INPUT_DATA0 on mismatch

- 4746 g. Verify that `inputData0 > 0`, and return error `TPM_DAA_INPUT_DATA0` on mismatch
- 4747 h. Set `DAA_tpmSpecific -> DAA_count = inputData0`
- 4748 i. set `DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific ||`
4749 `DAA_joinSession)`
- 4750 j. set `DAA_session -> DAA_stage = 1`
- 4751 k. Assign session handle for `TPM_DAA_Join`
- 4752 l. set `outputData = new session handle`
 - 4753 i. The handle in `outputData` is included the output HMAC.
- 4754 m. return `TPM_SUCCESS`
- 4755 1. If `stage==1`
 - 4756 a. Verify that `DAA_session ->DAA_stage==1`. Return `TPM_DAA_STAGE` and flush handle
4757 on mismatch
 - 4758 b. Verify that `DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||`
4759 `DAA_joinSession)` and return `TPM_DAA_TPM_SETTINGS` on mismatch
 - 4760 c. Verify that `sizeof(inputData0) == DAA_SIZE_issuerModulus` and return error
4761 `TPM_DAA_INPUT_DATA0` on mismatch
 - 4762 d. If `DAA_session -> DAA_scratch == NULL`:
 - 4763 i. Set `DAA_session -> DAA_scratch = inputData0`
 - 4764 ii. set `DAA_joinSession -> DAA_digest_n0 = SHA-1(DAA_session -> DAA_scratch)`
 - 4765 iii. set `DAA_tpmSpecific -> DAA_rekey = SHA-1(tpmDAASeed || DAA_joinSession ->`
4766 `DAA_digest_n0)`
 - 4767 e. Else (If `DAA_session -> DAA_scratch != NULL`):
 - 4768 i. Set `signedData = inputData0`
 - 4769 ii. Verify that `sizeof(inputData1) == DAA_SIZE_issuerModulus` and return error
4770 `TPM_DAA_INPUT_DATA1` on mismatch
 - 4771 iii. Set `signatureValue = inputData1`
 - 4772 iv. Use the RSA key `== [DAA_session -> DAA_scratch]` to verify that `signatureValue` is
4773 a signature on `signedData` using `TPM_SS_RSASSAPKCS1v15_SHA1` (RSA
4774 PKCS1.5 with SHA-1), and return error `TPM_DAA_ISSUER_VALIDITY` on
4775 mismatch
 - 4776 v. Set `DAA_session -> DAA_scratch = signedData`
 - 4777 f. Decrement `DAA_tpmSpecific -> DAA_count` by 1 (unity)
 - 4778 g. If `DAA_tpmSpecific -> DAA_count ==0`:
 - 4779 h. increment `DAA_session -> DAA_stage` by 1
 - 4780 i. set `DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific ||`
4781 `DAA_joinSession)`
 - 4782 j. set `outputData = NULL`

4783 k. return TPM_SUCCESS

4784 2. If stage==2

4785 a. Verify that DAA_session ->DAA_stage==2. Return TPM_DAA_STAGE and flush handle
4786 on mismatch

4787 b. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
4788 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

4789 c. Verify that sizeof(inputData0) == sizeof(TPM_DAA_ISSUER) and return error
4790 TPM_DAA_INPUT_DATA0 on mismatch

4791 d. Set DAA_issuerSettings = inputData0. Verify that all fields in DAA_issuerSettings are
4792 present and return error TPM_DAA_INPUT_DATA0 if not.

4793 e. Verify that sizeof(inputData1) == DAA_SIZE_issuerModulus and return error
4794 TPM_DAA_INPUT_DATA1 on mismatch

4795 f. Set signatureValue = inputData1

4796 g. Set signedData = (DAA_joinSession -> DAA_digest_n0 || DAA_issuerSettings)

4797 h. Use the RSA key [DAA_session -> DAA_scratch] to verify that signatureValue is a
4798 signature on signedData using TPM_SS_RSASSAPKCS1v15_SHA1 (RSA PKCS1.5 with
4799 SHA-1),, and return error TPM_DAA_ISSUER_VALIDITY on mismatch

4800 i. Set DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)

4801 j. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific ||
4802 DAA_joinSession)

4803 k. Set DAA_session -> DAA_scratch = NULL

4804 l. increment DAA_session -> DAA_stage by 1

4805 m. return TPM_SUCCESS

4806 3. If stage==3

4807 a. Verify that DAA_session ->DAA_stage==3. Return TPM_DAA_STAGE and flush handle
4808 on mismatch

4809 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
4810 return error TPM_DAA_ISSUER_SETTINGS on mismatch

4811 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
4812 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

4813 d. Verify that sizeof(inputData0) == sizeof(DAA_tpmSpecific -> DAA_count) and return
4814 error TPM_DAA_INPUT_DATA0 on mismatch

4815 e. Set DAA_tpmSpecific -> DAA_count = inputData0

4816 f. Obtain random data from the RNG and store it as DAA_joinSession -> DAA_join_u0

4817 g. Obtain random data from the RNG and store it as DAA_joinSession -> DAA_join_u1

4818 h. set outputData = NULL

4819 i. increment DAA_session -> DAA_stage by 1

4820 j. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific ||
4821 DAA_joinSession)

4822 k. return TPM_SUCCESS

4823 4. If stage==4,

4824 a. Verify that DAA_session ->DAA_stage==4. Return TPM_DAA_STAGE and flush handle
4825 on mismatch

4826 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
4827 return error TPM_DAA_ISSUER_SETTINGS on mismatch

4828 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
4829 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

4830 d. Set DAA_generic_R0 = inputData0

4831 e. Verify that SHA-1(DAA_generic_R0) == DAA_issuerSettings -> DAA_digest_R0 and
4832 return error TPM_DAA_INPUT_DATA0 on mismatch

4833 f. Set DAA_generic_n = inputData1

4834 g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and
4835 return error TPM_DAA_INPUT_DATA1 on mismatch

4836 h. Set X = DAA_generic_R0

4837 i. Set n = DAA_generic_n

4838 j. Set f = SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count ||
4839 0) || SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1)
4840 mod DAA_issuerSettings -> DAA_generic_q

4841 k. Set f0 = f mod 2^DAA_power0 (erase all but the lowest DAA_power0 bits of f)

4842 l. Set DAA_session -> DAA_scratch = (X^f0) mod n

4843 m. set outputData = NULL

4844 n. increment DAA_session -> DAA_stage by 1

4845 o. return TPM_SUCCESS

4846 5. If stage==5

4847 a. Verify that DAA_session ->DAA_stage==5. Return TPM_DAA_STAGE and flush handle
4848 on mismatch

4849 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
4850 return error TPM_DAA_ISSUER_SETTINGS on mismatch

4851 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
4852 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

4853 d. Set DAA_generic_R1 = inputData0

4854 e. Verify that SHA-1(DAA_generic_R1) == DAA_issuerSettings -> DAA_digest_R1 and
4855 return error TPM_DAA_INPUT_DATA0 on mismatch

4856 f. Set DAA_generic_n = inputData1

4857 g. Verify that $\text{SHA-1}(\text{DAA_generic_n}) == \text{DAA_issuerSettings} \rightarrow \text{DAA_digest_n}$ and
4858 return error `TPM_DAA_INPUT_DATA1` on mismatch

4859 h. Set $X = \text{DAA_generic_R1}$

4860 i. Set $n = \text{DAA_generic_n}$

4861 j. Set $f = \text{SHA-1}(\text{DAA_tpmSpecific} \rightarrow \text{DAA_rekey} || \text{DAA_tpmSpecific} \rightarrow \text{DAA_count} ||$
4862 $0) || \text{SHA-1}(\text{DAA_tpmSpecific} \rightarrow \text{DAA_rekey} || \text{DAA_tpmSpecific} \rightarrow \text{DAA_count} || 1)$
4863 $\text{mod DAA_issuerSettings} \rightarrow \text{DAA_generic_q}$.

4864 k. Shift f right by DAA_power0 bits (discard the lowest DAA_power0 bits) and label the
4865 result $f1$

4866 l. Set $Z = \text{DAA_session} \rightarrow \text{DAA_scratch}$

4867 m. Set $\text{DAA_session} \rightarrow \text{DAA_scratch} = Z \cdot (X^{f1}) \text{ mod } n$

4868 n. set `outputData = NULL`

4869 o. increment $\text{DAA_session} \rightarrow \text{DAA_stage}$ by 1

4870 p. return `TPM_SUCCESS`

4871 6. If `stage==6`

4872 a. Verify that $\text{DAA_session} \rightarrow \text{DAA_stage}==6$. Return `TPM_DAA_STAGE` and flush handle
4873 on mismatch

4874 b. Verify that $\text{DAA_tpmSpecific} \rightarrow \text{DAA_digestIssuer} == \text{SHA-1}(\text{DAA_issuerSettings})$ and
4875 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

4876 c. Verify that $\text{DAA_session} \rightarrow \text{DAA_digestContext} == \text{SHA-1}(\text{DAA_tpmSpecific} ||$
4877 $\text{DAA_joinSession})$ and return error `TPM_DAA_TPM_SETTINGS` on mismatch

4878 d. Set $\text{DAA_generic_S0} = \text{inputData0}$

4879 e. Verify that $\text{SHA-1}(\text{DAA_generic_S0}) == \text{DAA_issuerSettings} \rightarrow \text{DAA_digest_S0}$ and
4880 return error `TPM_DAA_INPUT_DATA0` on mismatch

4881 f. Set $\text{DAA_generic_n} = \text{inputData1}$

4882 g. Verify that $\text{SHA-1}(\text{DAA_generic_n}) == \text{DAA_issuerSettings} \rightarrow \text{DAA_digest_n}$ and
4883 return error `TPM_DAA_INPUT_DATA1` on mismatch

4884 h. Set $X = \text{DAA_generic_S0}$

4885 i. Set $n = \text{DAA_generic_n}$

4886 j. Set $Z = \text{DAA_session} \rightarrow \text{DAA_scratch}$

4887 k. Set $Y = \text{DAA_joinSession} \rightarrow \text{DAA_join_u0}$

4888 l. Set $\text{DAA_session} \rightarrow \text{DAA_scratch} = Z \cdot (X^Y) \text{ mod } n$

4889 m. set `outputData = NULL`

4890 n. increment $\text{DAA_session} \rightarrow \text{DAA_stage}$ by 1

4891 o. return `TPM_SUCCESS`

4892 7. If `stage==7`

- 4893 a. Verify that `DAA_session ->DAA_stage==7`. Return `TPM_DAA_STAGE` and flush handle
4894 on mismatch
- 4895 b. Verify that `DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and
4896 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch
- 4897 c. Verify that `DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||`
4898 `DAA_joinSession)` and return error `TPM_DAA_TPM_SETTINGS` on mismatch
- 4899 d. Set `DAA_generic_S1 = inputData0`
- 4900 e. Verify that `SHA-1(DAA_generic_S1) == DAA_issuerSettings -> DAA_digest_S1` and
4901 return error `TPM_DAA_INPUT_DATA0` on mismatch
- 4902 f. Set `DAA_generic_n = inputData1`
- 4903 g. Verify that `SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n` and
4904 return error `TPM_DAA_INPUT_DATA1` on mismatch
- 4905 h. Set `X = DAA_generic_S1`
- 4906 i. Set `n = DAA_generic_n`
- 4907 j. Set `Y = DAA_joinSession -> DAA_join_u1`
- 4908 k. Set `Z = DAA_session -> DAA_scratch`
- 4909 l. Set `DAA_session -> DAA_scratch = Z*(X^Y) mod n`
- 4910 m. Set `DAA_session -> DAA_digest` to the `SHA-1 (DAA_session -> DAA_scratch ||`
4911 `DAA_tpmSpecific -> DAA_count || DAA_joinSession -> DAA_digest_n0)`
- 4912 n. set `outputData = DAA_session -> DAA_scratch`
- 4913 o. set `DAA_session -> DAA_scratch = NULL`
- 4914 p. increment `DAA_session -> DAA_stage` by 1
- 4915 q. return `TPM_SUCCESS`
- 4916 8. If `stage==8`
- 4917 a. Verify that `DAA_session ->DAA_stage==8`. Return `TPM_DAA_STAGE` and flush handle
4918 on mismatch
- 4919 b. Verify that `DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and
4920 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch
- 4921 c. Verify that `DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||`
4922 `DAA_joinSession)` and return error `TPM_DAA_TPM_SETTINGS` on mismatch
- 4923 d. Verify `inputSize0 == DAA_SIZE_NE` and return error `TPM_DAA_INPUT_DATA0` on
4924 mismatch
- 4925 e. Set `NE = decrypt(inputData0, privEK)`
- 4926 f. set `outputData = SHA-1(DAA_session -> DAA_digest || NE)`
- 4927 g. set `DAA_session -> DAA_digest = NULL`
- 4928 h. increment `DAA_session -> DAA_stage` by 1
- 4929 i. return `TPM_SUCCESS`

- 4930 9. If stage==9
- 4931 a. Verify that DAA_session ->DAA_stage==9. Return TPM_DAA_STAGE and flush handle
4932 on mismatch
- 4933 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
4934 return error TPM_DAA_ISSUER_SETTINGS on mismatch
- 4935 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
4936 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
- 4937 d. Set DAA_generic_R0 = inputData0
- 4938 e. Verify that SHA-1(DAA_generic_R0) == DAA_issuerSettings -> DAA_digest_R0 and
4939 return error TPM_DAA_INPUT_DATA0 on mismatch
- 4940 f. Set DAA_generic_n = inputData1
- 4941 g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and
4942 return error TPM_DAA_INPUT_DATA1 on mismatch
- 4943 h. Obtain random data from the RNG and store it as DAA_session -> DAA_contextSeed
- 4944 i. Obtain DAA_SIZE_r0 bytes using the MGF1 function and label them Y. “r0” ||
4945 DAA_session -> DAA_contextSeed is the Z seed.
- 4946 j. Set X = DAA_generic_R0
- 4947 k. Set n = DAA_generic_n
- 4948 l. Set DAA_session -> DAA_scratch = (X^Y) mod n
- 4949 m. set outputData = NULL
- 4950 n. increment DAA_session -> DAA_stage by 1
- 4951 o. return TPM_SUCCESS
- 4952 10.If stage==10
- 4953 a. Verify that DAA_session ->DAA_stage==10. Return TPM_DAA_STAGE and flush
4954 handle on mismatch h
- 4955 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
4956 return error TPM_DAA_ISSUER_SETTINGS on mismatch
- 4957 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
4958 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
- 4959 d. Set DAA_generic_R1 = inputData0
- 4960 e. Verify that SHA-1(DAA_generic_R1) == DAA_issuerSettings -> DAA_digest_R1 and
4961 return error TPM_DAA_INPUT_DATA0 on mismatch
- 4962 f. Set DAA_generic_n = inputData1
- 4963 g. Verify that SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n and
4964 return error TPM_DAA_INPUT_DATA1 on mismatch
- 4965 h. Obtain DAA_SIZE_r1 bytes using the MGF1 function and label them Y. “r1” ||
4966 DAA_session -> DAA_contextSeed is the Z seed.
- 4967 i. Set X = DAA_generic_R1

- 4968 j. Set $n = \text{DAA_generic_n}$
- 4969 k. Set $Z = \text{DAA_session} \rightarrow \text{DAA_scratch}$
- 4970 l. Set $\text{DAA_session} \rightarrow \text{DAA_scratch} = Z \cdot (X^Y) \pmod n$
- 4971 m. set `outputData = NULL`
- 4972 n. increment `DAA_session` \rightarrow `DAA_stage` by 1
- 4973 o. return `TPM_SUCCESS`
- 4974 11.If `stage==11`
 - 4975 a. Verify that `DAA_session` \rightarrow `DAA_stage==11`. Return `TPM_DAA_STAGE` and flush
 - 4976 handle on mismatch
 - 4977 b. Verify that `DAA_tpmSpecific` \rightarrow `DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and
 - 4978 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch
 - 4979 c. Verify that `DAA_session` \rightarrow `DAA_digestContext == SHA-1(DAA_tpmSpecific ||`
 - 4980 `DAA_joinSession)` and return error `TPM_DAA_TPM_SETTINGS` on mismatch
 - 4981 d. Set `DAA_generic_S0 = inputData0`
 - 4982 e. Verify that `SHA-1(DAA_generic_S0) == DAA_issuerSettings` \rightarrow `DAA_digest_S0` and
 - 4983 return error `TPM_DAA_INPUT_DATA0` on mismatch
 - 4984 f. Set `DAA_generic_n = inputData1`
 - 4985 g. Verify that `SHA-1(DAA_generic_n) == DAA_issuerSettings` \rightarrow `DAA_digest_n` and
 - 4986 return error `TPM_DAA_INPUT_DATA1` on mismatch
 - 4987 h. Obtain `DAA_SIZE_r2` bytes using the MGF1 function and label them Y. “r2” ||
 - 4988 `DAA_session` \rightarrow `DAA_contextSeed` is the Z seed.
 - 4989 i. Set $X = \text{DAA_generic_S0}$
 - 4990 j. Set $n = \text{DAA_generic_n}$
 - 4991 k. Set $Z = \text{DAA_session} \rightarrow \text{DAA_scratch}$
 - 4992 l. Set $\text{DAA_session} \rightarrow \text{DAA_scratch} = Z \cdot (X^Y) \pmod n$
 - 4993 m. set `outputData = NULL`
 - 4994 n. increment `DAA_session` \rightarrow `DAA_stage` by 1
 - 4995 o. return `TPM_SUCCESS`
- 4996 12.If `stage==12`
 - 4997 a. Verify that `DAA_session` \rightarrow `DAA_stage==12`. Return `TPM_DAA_STAGE` and flush
 - 4998 handle on mismatch
 - 4999 b. Verify that `DAA_tpmSpecific` \rightarrow `DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and
 - 5000 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch
 - 5001 c. Verify that `DAA_session` \rightarrow `DAA_digestContext == SHA-1(DAA_tpmSpecific ||`
 - 5002 `DAA_joinSession)` and return error `TPM_DAA_TPM_SETTINGS` on mismatch
 - 5003 d. Set `DAA_generic_S1 = inputData0`

5004 e. Verify that $\text{SHA-1}(\text{DAA_generic_S1}) == \text{DAA_issuerSettings} \rightarrow \text{DAA_digest_S1}$ and
5005 return error `TPM_DAA_INPUT_DATA0` on mismatch

5006 f. Set `DAA_generic_n = inputData1`

5007 g. Verify that $\text{SHA-1}(\text{DAA_generic_n}) == \text{DAA_issuerSettings} \rightarrow \text{DAA_digest_n}$ and
5008 return error `TPM_DAA_INPUT_DATA1` on mismatch

5009 h. Obtain `DAA_SIZE_r3` bytes using the MGF1 function and label them Y. “r3” ||
5010 `DAA_session` \rightarrow `DAA_contextSeed` is the Z seed.

5011 i. Set `X = DAA_generic_S1`

5012 j. Set `n = DAA_generic_n`

5013 k. Set `Z = DAA_session` \rightarrow `DAA_scratch`

5014 l. Set `DAA_session` \rightarrow `DAA_scratch = Z*(X^Y) mod n`

5015 m. set `outputData = DAA_session` \rightarrow `DAA_scratch`

5016 n. Set `DAA_session` \rightarrow `DAA_scratch = NULL`

5017 o. increment `DAA_session` \rightarrow `DAA_stage` by 1

5018 p. return `TPM_SUCCESS`

5019 13.If `stage==13`

5020 a. Verify that `DAA_session` \rightarrow `DAA_stage==13`. Return `TPM_DAA_STAGE` and flush
5021 handle on mismatch

5022 b. Verify that `DAA_tpmSpecific` \rightarrow `DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and
5023 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5024 c. Verify that `DAA_session` \rightarrow `DAA_digestContext == SHA-1(DAA_tpmSpecific ||`
5025 `DAA_joinSession)` and return error `TPM_DAA_TPM_SETTINGS` on mismatch

5026 d. Set `DAA_generic_gamma = inputData0`

5027 e. Verify that $\text{SHA-1}(\text{DAA_generic_gamma}) == \text{DAA_issuerSettings} \rightarrow$
5028 `DAA_digest_gamma` and return error `TPM_DAA_INPUT_DATA0` on mismatch

5029 f. Verify that `inputSize1 == DAA_SIZE_w` and return error `TPM_DAA_INPUT_DATA1` on
5030 mismatch

5031 g. Set `w = inputData1`

5032 h. Set `w1 = w^(DAA_issuerSettings` \rightarrow `DAA_generic_q) mod (DAA_generic_gamma)`

5033 i. If `w1 != 1` (unity), return error `TPM_DAA_WRONG_W`

5034 j. Set `DAA_session` \rightarrow `DAA_scratch = w`

5035 k. set `outputData = NULL`

5036 l. increment `DAA_session` \rightarrow `DAA_stage` by 1

5037 m. return `TPM_SUCCESS`.

5038 14.If `stage==14`

5039 a. Verify that `DAA_session` \rightarrow `DAA_stage==14`. Return `TPM_DAA_STAGE` and flush
5040 handle on mismatch

5041 b. Verify that $DAA_tpmSpecific \rightarrow DAA_digestIssuer == SHA-1(DAA_issuerSettings)$ and
5042 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5043 c. Verify that $DAA_session \rightarrow DAA_digestContext == SHA-1(DAA_tpmSpecific ||$
5044 $DAA_joinSession)$ and return error `TPM_DAA_TPM_SETTINGS` on mismatch

5045 d. Set $DAA_generic_gamma = inputData0$

5046 e. Verify that $SHA-1(DAA_generic_gamma) == DAA_issuerSettings \rightarrow$
5047 DAA_digest_gamma and return error `TPM_DAA_INPUT_DATA0` on mismatch

5048 f. Set $f = SHA-1(DAA_tpmSpecific \rightarrow DAA_rekey || DAA_tpmSpecific \rightarrow DAA_count ||$
5049 $0) || SHA-1(DAA_tpmSpecific \rightarrow DAA_rekey || DAA_tpmSpecific \rightarrow DAA_count || 1)$
5050 $\text{mod } DAA_issuerSettings \rightarrow DAA_generic_q$.

5051 g. Set $E = ((DAA_session \rightarrow DAA_scratch)^f) \text{ mod } (DAA_generic_gamma)$.

5052 h. Set $outputData = E$

5053 i. increment $DAA_session \rightarrow DAA_stage$ by 1

5054 j. return `TPM_SUCCESS`.

5055 15.If $stage==15$

5056 a. Verify that $DAA_session \rightarrow DAA_stage==15$. Return `TPM_DAA_STAGE` and flush
5057 handle on mismatch

5058 b. Verify that $DAA_tpmSpecific \rightarrow DAA_digestIssuer == SHA-1(DAA_issuerSettings)$ and
5059 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5060 c. Verify that $DAA_session \rightarrow DAA_digestContext == SHA-1(DAA_tpmSpecific ||$
5061 $DAA_joinSession)$ and return error `TPM_DAA_TPM_SETTINGS` on mismatch

5062 d. Set $DAA_generic_gamma = inputData0$

5063 e. Verify that $SHA-1(DAA_generic_gamma) == DAA_issuerSettings \rightarrow$
5064 DAA_digest_gamma and return error `TPM_DAA_INPUT_DATA0` on mismatch

5065 f. Obtain DAA_SIZE_r0 bytes using the MGF1 function and label them $r0$. " $r0$ " ||
5066 $DAA_session \rightarrow DAA_contextSeed$ is the Z seed.

5067 g. Obtain DAA_SIZE_r1 bytes using the MGF1 function and label them $r1$. " $r1$ " ||
5068 $DAA_session \rightarrow DAA_contextSeed$ is the Z seed.

5069 h. set $r = r0 + 2^{DAA_power0} * r1 \text{ mod } (DAA_issuerSettings \rightarrow DAA_generic_q)$.

5070 i. set $E1 = ((DAA_session \rightarrow DAA_scratch)^r) \text{ mod } (DAA_generic_gamma)$.

5071 j. Set $DAA_session \rightarrow DAA_scratch = NULL$

5072 k. Set $outputData = E1$

5073 l. increment $DAA_session \rightarrow DAA_stage$ by 1

5074 m. return `TPM_SUCCESS`.

5075 16.If $stage==16$

5076 a. Verify that $DAA_session \rightarrow DAA_stage==16$. Return `TPM_DAA_STAGE` and flush
5077 handle on mismatch

5078 b. Verify that `DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and
5079 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5080 c. Verify that `DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||`
5081 `DAA_joinSession)` and return error `TPM_DAA_TPM_SETTINGS` on mismatch

5082 d. Verify that `inputSize0 == sizeof(TPM_DIGEST)` and return error
5083 `TPM_DAA_INPUT_DATA0` on mismatch

5084 e. Set `DAA_session -> DAA_digest = inputData0`

5085 f. Obtain `DAA_SIZE_NT` bytes from the RNG and label them NT

5086 g. Set `DAA_session -> DAA_digest` to the SHA-1 (`DAA_session -> DAA_digest || NT`)

5087 h. Set `outputData = NT`

5088 i. increment `DAA_session -> DAA_stage` by 1

5089 j. return `TPM_SUCCESS`.

5090 17.If `stage==17`

5091 a. Verify that `DAA_session ->DAA_stage==17`. Return `TPM_DAA_STAGE` and flush
5092 handle on mismatch

5093 b. Verify that `DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and
5094 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5095 c. Verify that `DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||`
5096 `DAA_joinSession)` and return error `TPM_DAA_TPM_SETTINGS` on mismatch

5097 d. Obtain `DAA_SIZE_r0` bytes using the MGF1 function and label them r0. “r0” ||
5098 `DAA_session -> DAA_contextSeed` is the Z seed.

5099 e. Set `f = SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count ||`
5100 `0) || SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1)`
5101 `mod DAA_issuerSettings -> DAA_generic_q`.

5102 f. Set `f0 = f mod 2^DAA_power0` (erase all but the lowest `DAA_power0` bits of f)

5103 g. Set `s0 = r0 + (DAA_session -> DAA_digest) * f0` in **Z**. Compute over the integers. The
5104 computation is not reduced with a modulus.

5105 h. set `outputData = s0`

5106 i. increment `DAA_session -> DAA_stage` by 1

5107 j. return `TPM_SUCCESS`

5108 18.If `stage==18`

5109 a. Verify that `DAA_session ->DAA_stage==18`. Return `TPM_DAA_STAGE` and flush
5110 handle on mismatch

5111 b. Verify that `DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and
5112 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5113 c. Verify that `DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||`
5114 `DAA_joinSession)` and return error `TPM_DAA_TPM_SETTINGS` on mismatch

5115 d. Obtain `DAA_SIZE_r1` bytes using the MGF1 function and label them r1. “r1” ||
5116 `DAA_session -> DAA_contextSeed` is the Z seed.

5117 e. Set $f = \text{SHA-1}(\text{DAA_tpmSpecific} \rightarrow \text{DAA_rekey} \parallel \text{DAA_tpmSpecific} \rightarrow \text{DAA_count} \parallel$
5118 $0) \parallel \text{SHA-1}(\text{DAA_tpmSpecific} \rightarrow \text{DAA_rekey} \parallel \text{DAA_tpmSpecific} \rightarrow \text{DAA_count} \parallel 1)$
5119 $\text{mod } \text{DAA_issuerSettings} \rightarrow \text{DAA_generic_q}.$

5120 f. Shift f right by DAA_power0 bits (discard the lowest DAA_power0 bits) and label the
5121 result $f1$

5122 g. Set $s1 = r1 + (\text{DAA_session} \rightarrow \text{DAA_digest}) * f1$ in **Z**. Compute over the integers. The
5123 computation is not reduced with a modulus.

5124 h. set $\text{outputData} = s1$

5125 i. increment $\text{DAA_session} \rightarrow \text{DAA_stage}$ by 1

5126 j. return `TPM_SUCCESS`

5127 19.If $\text{stage}==19$

5128 a. Verify that $\text{DAA_session} \rightarrow \text{DAA_stage}==19$. Return `TPM_DAA_STAGE` and flush
5129 handle on mismatch

5130 b. Verify that $\text{DAA_tpmSpecific} \rightarrow \text{DAA_digestIssuer} == \text{SHA-1}(\text{DAA_issuerSettings})$ and
5131 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5132 c. Verify that $\text{DAA_session} \rightarrow \text{DAA_digestContext} == \text{SHA-1}(\text{DAA_tpmSpecific} \parallel$
5133 $\text{DAA_joinSession})$ and return error `TPM_DAA_TPM_SETTINGS` on mismatch

5134 d. Obtain DAA_SIZE_r2 bytes using the MGF1 function and label them $r2$. " $r2$ " \parallel
5135 $\text{DAA_session} \rightarrow \text{DAA_contextSeed}$ is the Z seed.

5136 e. Set $s2 = r2 + (\text{DAA_session} \rightarrow \text{DAA_digest}) * (\text{DAA_joinSession} \rightarrow \text{DAA_join_u0}) \text{ mod}$
5137 $2^{\text{DAA_power1}}$ (Erase all but the lowest DAA_power1 bits of $s2$)

5138 f. set $\text{outputData} = s2$

5139 g. increment $\text{DAA_session} \rightarrow \text{DAA_stage}$ by 1

5140 h. return `TPM_SUCCESS`

5141 20.If $\text{stage}==20$

5142 a. Verify that $\text{DAA_session} \rightarrow \text{DAA_stage}==20$. Return `TPM_DAA_STAGE` and flush
5143 handle on mismatch

5144 b. Verify that $\text{DAA_tpmSpecific} \rightarrow \text{DAA_digestIssuer} == \text{SHA-1}(\text{DAA_issuerSettings})$ and
5145 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5146 c. Verify that $\text{DAA_session} \rightarrow \text{DAA_digestContext} == \text{SHA-1}(\text{DAA_tpmSpecific} \parallel$
5147 $\text{DAA_joinSession})$ and return error `TPM_DAA_TPM_SETTINGS` on mismatch

5148 d. Obtain DAA_SIZE_r2 bytes using the MGF1 function and label them $r2$. " $r2$ " \parallel
5149 $\text{DAA_session} \rightarrow \text{DAA_contextSeed}$ is the Z seed.

5150 e. Set $s12 = r2 + (\text{DAA_session} \rightarrow \text{DAA_digest}) * (\text{DAA_joinSession} \rightarrow \text{DAA_join_u0})$

5151 f. Shift $s12$ right by DAA_power1 bit (discard the lowest DAA_power1 bits).

5152 g. Set $\text{DAA_session} \rightarrow \text{DAA_scratch} = s12$

5153 h. Set $\text{outputData} = \text{DAA_session} \rightarrow \text{DAA_digest}$

5154 i. increment $\text{DAA_session} \rightarrow \text{DAA_stage}$ by 1

```

5155     j. return TPM_SUCCESS
5156 21.If stage==21
5157     a. Verify that DAA_session ->DAA_stage==21. Return TPM_DAA_STAGE and flush
5158     handle on mismatch
5159     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5160     return error TPM_DAA_ISSUER_SETTINGS on mismatch
5161     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5162     DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
5163     d. Obtain DAA_SIZE_r3 bytes using the MGF1 function and label them r3. “r3” ||
5164     DAA_session -> DAA_contextSeed is the Z seed.
5165     e. Set s3 = r3 + (DAA_session -> DAA_digest)*( DAA_joinSession -> DAA_join_u1) +
5166     (DAA_session -> DAA_scratch).
5167     f. Set DAA_session -> DAA_scratch = NULL
5168     g. set outputData = s3
5169     h. increment DAA_session -> DAA_stage by 1
5170     i. return TPM_SUCCESS
5171 22.If stage==22
5172     a. Verify that DAA_session ->DAA_stage==22. Return TPM_DAA_STAGE and flush
5173     handle on mismatch
5174     b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5175     return error TPM_DAA_ISSUER_SETTINGS on mismatch
5176     c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5177     DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
5178     d. Verify inputSize0 == DAA_SIZE_v0 and return error TPM_DAA_INPUT_DATA0 on
5179     mismatch
5180     e. Set u2 = inputData0
5181     f. Set v0 = u2 + (DAA_joinSession -> DAA_join_u0) mod 2^DAA_power1 (Erase all but
5182     the lowest DAA_power1 bits of v0).
5183     g. Set DAA_tpmSpecific -> DAA_digest_v0 = SHA-1(v0)
5184     h. Set v10 = u2 + (DAA_joinSession -> DAA_join_u0) in Z. Compute over the integers.
5185     The computation is not reduced with a modulus.
5186     i. Shift v10 right by DAA_power1 bits (erase the lowest DAA_power1 bits).
5187     j. Set DAA_session ->DAA_scratch = v10
5188     k. Set outputData
5189         i. Fill in TPM_DAA_BLOB with a type of TPM_RT_DAA_V0 and encrypt the v0
5190         parameters using TPM_PERMANENT_DATA -> daaBlobKey
5191         ii. set outputData to the encrypted TPM_DAA_BLOB
5192     l. increment DAA_session -> DAA_stage by 1

```

5193 m. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific ||
5194 DAA_joinSession)

5195 n. return TPM_SUCCESS

5196 23.If stage==23

5197 a. Verify that DAA_session ->DAA_stage==23. Return TPM_DAA_STAGE and flush
5198 handle on mismatch

5199 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5200 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5201 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5202 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

5203 d. Verify inputSize0 == DAA_SIZE_v1 and return error TPM_DAA_INPUT_DATA0 on
5204 mismatch

5205 e. Set u3 = inputData0

5206 f. Set v1 = u3 + DAA_joinSession -> DAA_join_u1 + DAA_session ->DAA_scratch

5207 g. Set DAA_tpmSpecific -> DAA_digest_v1 = SHA-1(v1)

5208 h. Set outputData

5209 i. Fill in TPM_DAA_BLOB with a type of TPM_RT_DAA_V1 and encrypt the v1
5210 parameters using TPM_PERMANENT_DATA -> daaBlobKey

5211 ii. set outputData to the encrypted TPM_DAA_BLOB

5212 i. Set DAA_session ->DAA_scratch = NULL

5213 j. increment DAA_session -> DAA_stage by 1

5214 k. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific ||
5215 DAA_joinSession)

5216 l. return TPM_SUCCESS

5217 24.If stage==24

5218 a. Verify that DAA_session ->DAA_stage==24. Return TPM_DAA_STAGE and flush
5219 handle on mismatch

5220 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5221 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5222 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5223 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

5224 d. set outputData = enc(DAA_tpmSpecific) using TPM_PERMANENT_DATA ->
5225 daaBlobKey

5226 e. Terminate the DAA session and all resources associated with the DAA join session
5227 handle.

5228 f. return TPM_SUCCESS

5229 25.If stage > 24, return error: TPM_DAA_STAGE

5230 **26.2 TPM_DAA_Sign**5231 **Start of informative comment:**

5232 outputSize and outputData are always included in the outParamDigest. This includes stage
5233 0, where the outputData contains the DAA session handle.

5234 **End of informative comment.**

5235 TPM protected capability; user must provide authorizations from the TPM Owner.

5236 **Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	Tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	Ordinal	Command ordinal: TPM_ORD_DAA_Sign
4	4			TPM_HANDLE	handle	Handle to the sign session
5	1	2S	1	BYTE	stage	Stage of the sign process
6	4	3S	4	UINT32	inputSize0	Size of inputData0 for this stage of DAA_Sign
7	<>	4S	<>	BYTE[]	inputData0	Data to be used by this capability
8	4	5S	4	UINT32	inputSize1	Size of inputData1 for this stage of DAA_Sign
9	<>	6S	<>	BYTE[]	inputData1	Data to be used by this capability
10	4			TPM_AUTHHANDLE	authHandle	The authorization 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.

5237 **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_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.

5238 **Description**

5239 This table summaries the input, output and saved data that is associated with each stage of
5240 processing.

Stage	Input Data0	Input Data1	Operation	Output Data	Scratchpad
0	DAA_issuerSettings	NULL	initialise	handle	NULL
1	enc(DAA_tpmSpecific)	NULL	initialise	NULL	NULL
2	DAA_generic_R0	DAA_generic_n	$P1 = R0^{r0} \text{ mod } n$	NULL	P1
3	DAA_generic_R1	DAA_generic_n	$P2 = P1 * (R1^{r1}) \text{ mod } n$	NULL	P2
4	DAA_generic_S0	DAA_generic_n	$P3 = P2 * (S0^{r2}) \text{ mod } n$	NULL	P3
5	DAA_generic_S1	DAA_generic_n	$T = P3 * (S1^{r4}) \text{ mod } n$	T	NULL
6	DAA_generic_gamma	w	$w1 = w^q \text{ mod } \text{gamma}$	NULL	w
7	DAA_generic_gamma	NULL	$E = w^f \text{ mod } \text{gamma}$	E	w
8	DAA_generic_gamma	NULL	$r = r0 + (2^{\text{power}0}) * r1 \text{ mod } q,$ $E1 = w^r \text{ mod } \text{gamma}$	E1	NULL
9	c1	NULL	$c = \text{hash}(c1 \parallel \text{NT})$	NT	NULL
10	b (selector)	m or handle to AIK	$c = \text{hash}(c \parallel 1 \parallel m)$ or $c = \text{hash}(c \parallel 0 \parallel \text{AIK-modulus})$	c	NULL
11	NULL	NULL	$s0 = r0 + c * f0$	s0	NULL
12	NULL	NULL	$s1 = r1 + c * f1$	s1	NULL
13	enc(v0)	NULL	$s2 = r2 + c * v0 \text{ mod } 2^{\text{power}1}$	s2	NULL
14	enc(v0)	NULL	$s12 = r2 + c * v0 \gg \text{power}1$	NULL	s12
15	enc(v1)	NULL	$s3 = r4 + c * v1 + s12$	s3	NULL

5241

5242 When a TPM receives an Owner authorized command to input enc(DAA_tpmSpecific) or
5243 enc(v0) or enc(v1), the TPM MUST verify that the TPM created the data and that neither the
5244 data nor the TPM's daaProof has been changed since the data was created. Loading one of
5245 these wrapped blobs does not require authorization, since correct blobs were created by the
5246 TPM under Owner authorization, and unwrapped blobs cannot be used without Owner
5247 authorisation. The TPM MUST NOT restrict the number of times that the contents of
5248 enc(DAA_tpmSpecific) or enc(v0) or enc(v1) can be used by the same combination of TPM
5249 and daaProof that created them.

5250 **Actions**

5251 A Trusted Platform Module that receives a valid TPM_DAA_Sign command SHALL:

- 5252 1. Use ownerAuth to verify that the Owner authorized all TPM_DAA_Sign input parameters.
- 5253 2. Any error results in the TPM invalidating all resources associated with the command
- 5254 3. Constant values of 0 or 1 are 1 byte integers, stages affected are
 - 5255 a. 7(f), 11(e), 12(e)

- 5256 4. Representation of the strings “r0” to “r4” are 2-byte ASCII encodings, stages affected are
5257 a. 2(h), 3(h), 4(h), 5(h), 12(d), 13(f), 14(f), 15(f)

5258 **Stages**

- 5259 0. If stage==0
- 5260 a. Determine that sufficient resources are available to perform a TPM_DAA_Sign.
 - 5261 i. The TPM MUST support sufficient resources to perform one (1) TPM_DAA_Join/
5262 TPM_DAA_Sign. The TPM MAY support addition TPM_DAA_Join/ TPM_DAA_Sign
5263 sessions.
 - 5264 ii. The TPM may share internal resources between the DAA operations and other
5265 variable resource requirements:
 - 5266 iii. If there are insufficient resources within the stored key pool (and one or more
5267 keys need to be removed to permit the DAA operation to execute) return
5268 TPM_NOSPACE
 - 5269 iv. If there are insufficient resources within the stored session pool (and one or
5270 more authorization or transport sessions need to be removed to permit the
5271 DAA operation to execute), return TPM_RESOURCES.
 - 5272 b. Set DAA_issuerSettings = inputData0
 - 5273 c. Verify that all fields in DAA_issuerSettings are present and return error
5274 TPM_DAA_INPUT_DATA0 if not.
 - 5275 d. set all fields in DAA_session = NULL
 - 5276 e. Assign new handle for session
 - 5277 f. Set outputData to new handle
 - 5278 i. The handle in outputData is included the output HMAC.
 - 5279 g. set DAA_session -> DAA_stage = 1
 - 5280 h. return TPM_SUCCESS
- 5281 1. If stage==1
- 5282 a. Verify that DAA_session ->DAA_stage==1. Return TPM_DAA_STAGE and flush handle
5283 on mismatch
 - 5284 b. Set DAA_tpmSpecific = unwrap(inputData0) using TPM_PERMANENT_DATA ->
5285 daaBlobKey
 - 5286 c. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5287 return error TPM_DAA_ISSUER_SETTINGS on mismatch
 - 5288 d. set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific)
 - 5289 e. set outputData = NULL
 - 5290 f. set DAA_session -> DAA_stage =2
 - 5291 g. return TPM_SUCCESS
- 5292 2. If stage==2

- 5293 a. Verify that `DAA_session ->DAA_stage==2`. Return `TPM_DAA_STAGE` and flush handle
5294 on mismatch
- 5295 b. Verify that `DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and
5296 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch
- 5297 c. Verify that `DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific)` and
5298 return error `TPM_DAA_TPM_SETTINGS` on mismatch
- 5299 d. Set `DAA_generic_R0 = inputData0`
- 5300 e. Verify that `SHA-1(DAA_generic_R0) == DAA_issuerSettings -> DAA_digest_R0` and
5301 return error `TPM_DAA_INPUT_DATA0` on mismatch
- 5302 f. Set `DAA_generic_n = inputData1`
- 5303 g. Verify that `SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n` and
5304 return error `TPM_DAA_INPUT_DATA1` on mismatch
- 5305 h. Obtain random data from the RNG and store it as `DAA_session -> DAA_contextSeed`
- 5306 i. Obtain `DAA_SIZE_r0` bytes using the MGF1 function and label them Y. “r0” ||
5307 `DAA_session -> DAA_contextSeed` is the Z seed.
- 5308 j. Set `X = DAA_generic_R0`
- 5309 k. Set `n = DAA_generic_n`
- 5310 l. Set `DAA_session -> DAA_scratch = (X^Y) mod n`
- 5311 m. set `outputData = NULL`
- 5312 n. increment `DAA_session -> DAA_stage` by 1
- 5313 o. return `TPM_SUCCESS`
- 5314 3. If `stage==3`
- 5315 a. Verify that `DAA_session ->DAA_stage==3`. Return `TPM_DAA_STAGE` and flush handle
5316 on mismatch
- 5317 b. Verify that `DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and
5318 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch
- 5319 c. Verify that `DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific)` and
5320 return error `TPM_DAA_TPM_SETTINGS` on mismatch
- 5321 d. Set `DAA_generic_R1 = inputData0`
- 5322 e. Verify that `SHA-1(DAA_generic_R1) == DAA_issuerSettings -> DAA_digest_R1` and
5323 return error `TPM_DAA_INPUT_DATA0` on mismatch
- 5324 f. Set `DAA_generic_n = inputData1`
- 5325 g. Verify that `SHA-1(DAA_generic_n) == DAA_issuerSettings -> DAA_digest_n` and
5326 return error `TPM_DAA_INPUT_DATA1` on mismatch
- 5327 h. Obtain `DAA_SIZE_r1` bytes using the MGF1 function and label them Y. “r1” ||
5328 `DAA_session -> DAA_contextSeed` is the Z seed.
- 5329 i. Set `X = DAA_generic_R1`
- 5330 j. Set `n = DAA_generic_n`

- 5331 k. Set $Z = \text{DAA_session} \rightarrow \text{DAA_scratch}$
- 5332 l. Set $\text{DAA_session} \rightarrow \text{DAA_scratch} = Z \cdot (X^Y) \pmod n$
- 5333 m. set `outputData = NULL`
- 5334 n. increment `DAA_session` \rightarrow `DAA_stage` by 1
- 5335 o. return `TPM_SUCCESS`
- 5336 4. If `stage==4`
- 5337 a. Verify that `DAA_session` \rightarrow `DAA_stage==4`. Return `TPM_DAA_STAGE` and flush handle
5338 on mismatch
- 5339 b. Verify that `DAA_tpmSpecific` \rightarrow `DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and
5340 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch
- 5341 c. Verify that `DAA_session` \rightarrow `DAA_digestContext = SHA-1(DAA_tpmSpecific)` and
5342 return error `TPM_DAA_TPM_SETTINGS` on mismatch
- 5343 d. Set `DAA_generic_S0 = inputData0`
- 5344 e. Verify that $\text{SHA-1}(\text{DAA_generic_S0}) == \text{DAA_issuerSettings} \rightarrow \text{DAA_digest_S0}$ and
5345 return error `TPM_DAA_INPUT_DATA0` on mismatch
- 5346 f. Set `DAA_generic_n = inputData1`
- 5347 g. Verify that $\text{SHA-1}(\text{DAA_generic_n}) == \text{DAA_issuerSettings} \rightarrow \text{DAA_digest_n}$ and
5348 return error `TPM_DAA_INPUT_DATA1` on mismatch
- 5349 h. Obtain `DAA_SIZE_r2` bytes using the MGF1 function and label them Y. “r2” ||
5350 `DAA_session` \rightarrow `DAA_contextSeed` is the Z seed.
- 5351 i. Set $X = \text{DAA_generic_S0}$
- 5352 j. Set $n = \text{DAA_generic_n}$
- 5353 k. Set $Z = \text{DAA_session} \rightarrow \text{DAA_scratch}$
- 5354 l. Set $\text{DAA_session} \rightarrow \text{DAA_scratch} = Z \cdot (X^Y) \pmod n$
- 5355 m. set `outputData = NULL`
- 5356 n. increment `DAA_session` \rightarrow `DAA_stage` by 1
- 5357 o. return `TPM_SUCCESS`
- 5358 5. If `stage==5`
- 5359 a. Verify that `DAA_session` \rightarrow `DAA_stage==5`. Return `TPM_DAA_STAGE` and flush handle
5360 on mismatch
- 5361 b. Verify that `DAA_tpmSpecific` \rightarrow `DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and
5362 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch
- 5363 c. Verify that `DAA_session` \rightarrow `DAA_digestContext == SHA-1(DAA_tpmSpecific)` and
5364 return error `TPM_DAA_TPM_SETTINGS` on mismatch
- 5365 d. Set `DAA_generic_S1 = inputData0`
- 5366 e. Verify that $\text{SHA-1}(\text{DAA_generic_S1}) == \text{DAA_issuerSettings} \rightarrow \text{DAA_digest_S1}$ and
5367 return error `TPM_DAA_INPUT_DATA0` on mismatch

- 5368 f. Set $DAA_generic_n = inputData1$
- 5369 g. Verify that $SHA-1(DAA_generic_n) == DAA_issuerSettings$ -> DAA_digest_n and
5370 return error $TPM_DAA_INPUT_DATA1$ on mismatch
- 5371 h. Obtain DAA_SIZE_r4 bytes using the MGF1 function and label them Y. “r4” ||
5372 $DAA_session$ -> $DAA_contextSeed$ is the Z seed.
- 5373 i. Set $X = DAA_generic_S1$
- 5374 j. Set $n = DAA_generic_n$
- 5375 k. Set $Z = DAA_session$ -> $DAA_scratch$
- 5376 l. Set $DAA_session$ -> $DAA_scratch = Z*(X^Y) \bmod n$
- 5377 m. set $outputData = DAA_session$ -> $DAA_scratch$
- 5378 n. set $DAA_session$ -> $DAA_scratch = NULL$
- 5379 o. increment $DAA_session$ -> DAA_stage by 1
- 5380 p. return $TPM_SUCCESS$
- 5381 6. If $stage==6$
- 5382 a. Verify that $DAA_session$ -> $DAA_stage==6$. Return TPM_DAA_STAGE and flush handle
5383 on mismatch
- 5384 b. Verify that $DAA_tpmSpecific$ -> $DAA_digestIssuer == SHA-1(DAA_issuerSettings)$ and
5385 return error $TPM_DAA_ISSUER_SETTINGS$ on mismatch
- 5386 c. Verify that $DAA_session$ -> $DAA_digestContext == SHA-1(DAA_tpmSpecific)$ and
5387 return error $TPM_DAA_TPM_SETTINGS$ on mismatch
- 5388 d. Set $DAA_generic_gamma = inputData0$
- 5389 e. Verify that $SHA-1(DAA_generic_gamma) == DAA_issuerSettings$ ->
5390 DAA_digest_gamma and return error $TPM_DAA_INPUT_DATA0$ on mismatch
- 5391 f. Verify that $inputSize1 == DAA_SIZE_w$ and return error $TPM_DAA_INPUT_DATA1$ on
5392 mismatch
- 5393 g. Set $w = inputData1$
- 5394 h. Set $w1 = w^{(DAA_issuerSettings -> DAA_generic_q)} \bmod (DAA_generic_gamma)$
- 5395 i. If $w1 \neq 1$ (unity), return error $TPM_DAA_WRONG_W$
- 5396 j. Set $DAA_session$ -> $DAA_scratch = w$
- 5397 k. set $outputData = NULL$
- 5398 l. increment $DAA_session$ -> DAA_stage by 1
- 5399 m. return $TPM_SUCCESS$.
- 5400 7. If $stage==7$
- 5401 a. Verify that $DAA_session$ -> $DAA_stage==7$. Return TPM_DAA_STAGE and flush handle
5402 on mismatch
- 5403 b. Verify that $DAA_tpmSpecific$ -> $DAA_digestIssuer == SHA-1(DAA_issuerSettings)$ and
5404 return error $TPM_DAA_ISSUER_SETTINGS$ on mismatch

5405 c. Verify that $\text{DAA_session} \rightarrow \text{DAA_digestContext} == \text{SHA-1}(\text{DAA_tpmSpecific})$ and
5406 return error `TPM_DAA_TPM_SETTINGS` on mismatch

5407 d. Set $\text{DAA_generic_gamma} = \text{inputData0}$

5408 e. Verify that $\text{SHA-1}(\text{DAA_generic_gamma}) == \text{DAA_issuerSettings} \rightarrow$
5409 DAA_digest_gamma and return error `TPM_DAA_INPUT_DATA0` on mismatch

5410 f. Set $f = \text{SHA-1}(\text{DAA_tpmSpecific} \rightarrow \text{DAA_rekey} || \text{DAA_tpmSpecific} \rightarrow \text{DAA_count} ||$
5411 $0) || \text{SHA-1}(\text{DAA_tpmSpecific} \rightarrow \text{DAA_rekey} || \text{DAA_tpmSpecific} \rightarrow \text{DAA_count} || 1)$
5412 $\text{mod DAA_issuerSettings} \rightarrow \text{DAA_generic_q}$.

5413 g. Set $E = ((\text{DAA_session} \rightarrow \text{DAA_scratch})^f) \text{ mod } (\text{DAA_generic_gamma})$.

5414 h. Set $\text{outputData} = E$

5415 i. increment $\text{DAA_session} \rightarrow \text{DAA_stage}$ by 1

5416 j. return `TPM_SUCCESS`.

5417 8. If $\text{stage} == 8$

5418 a. Verify that $\text{DAA_session} \rightarrow \text{DAA_stage} == 8$. Return `TPM_DAA_STAGE` and flush handle
5419 on mismatch

5420 b. Verify that $\text{DAA_tpmSpecific} \rightarrow \text{DAA_digestIssuer} == \text{SHA-1}(\text{DAA_issuerSettings})$ and
5421 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5422 c. Verify that $\text{DAA_session} \rightarrow \text{DAA_digestContext} == \text{SHA-1}(\text{DAA_tpmSpecific})$ and
5423 return error `TPM_DAA_TPM_SETTINGS` on mismatch

5424 d. Set $\text{DAA_generic_gamma} = \text{inputData0}$

5425 e. Verify that $\text{SHA-1}(\text{DAA_generic_gamma}) == \text{DAA_issuerSettings} \rightarrow$
5426 DAA_digest_gamma and return error `TPM_DAA_INPUT_DATA0` on mismatch

5427 f. Obtain DAA_SIZE_r0 bytes using the MGF1 function and label them r_0 . " r_0 " ||
5428 $\text{DAA_session} \rightarrow \text{DAA_contextSeed}$ is the Z seed.

5429 g. Obtain DAA_SIZE_r1 bytes using the MGF1 function and label them r_1 . " r_1 " ||
5430 $\text{DAA_session} \rightarrow \text{DAA_contextSeed}$ is the Z seed.

5431 h. set $r = r_0 + 2^{\text{DAA_power0}} * r_1 \text{ mod } (\text{DAA_issuerSettings} \rightarrow \text{DAA_generic_q})$.

5432 i. Set $E_1 = ((\text{DAA_session} \rightarrow \text{DAA_scratch})^r) \text{ mod } (\text{DAA_generic_gamma})$

5433 j. Set $\text{DAA_session} \rightarrow \text{DAA_scratch} = \text{NULL}$

5434 k. Set $\text{outputData} = E_1$

5435 l. increment $\text{DAA_session} \rightarrow \text{DAA_stage}$ by 1

5436 m. return `TPM_SUCCESS`.

5437 9. If $\text{stage} == 9$

5438 a. Verify that $\text{DAA_session} \rightarrow \text{DAA_stage} == 9$. Return `TPM_DAA_STAGE` and flush handle
5439 on mismatch

5440 b. Verify that $\text{DAA_tpmSpecific} \rightarrow \text{DAA_digestIssuer} == \text{SHA-1}(\text{DAA_issuerSettings})$ and
5441 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5442 c. Verify that `DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific)` and
5443 return error `TPM_DAA_TPM_SETTINGS` on mismatch

5444 d. Verify that `inputSize0 == sizeof(TPM_DIGEST)` and return error
5445 `TPM_DAA_INPUT_DATA0` on mismatch

5446 e. Set `DAA_session -> DAA_digest = inputData0`

5447 f. Obtain `DAA_SIZE_NT` bytes from the RNG and label them NT

5448 g. Set `DAA_session -> DAA_digest` to the SHA-1 (`DAA_session -> DAA_digest || NT`)

5449 h. Set `outputData = NT`

5450 i. increment `DAA_session -> DAA_stage` by 1

5451 j. return `TPM_SUCCESS`.

5452 10.If `stage==10`

5453 a. Verify that `DAA_session ->DAA_stage==10`. Return `TPM_DAA_STAGE` and flush
5454 handle on mismatch

5455 b. Verify that `DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and
5456 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5457 c. Verify that `DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific)` and
5458 return error `TPM_DAA_TPM_SETTINGS` on mismatch

5459 d. Verify that `inputSize0 == sizeof(BYTE)`, and return error `TPM_DAA_INPUT_DATA0` on
5460 mismatch

5461 e. Set `selector = inputData0`, verify that `selector == 0` or `1`, and return error
5462 `TPM_DAA_INPUT_DATA0` on mismatch

5463 f. If `selector == 1`, verify that `inputSize1 == sizeof(TPM_DIGEST)`, and return error
5464 `TPM_DAA_INPUT_DATA1` on mismatch

5465 g. Set `DAA_session -> DAA_digest` to SHA-1 (`DAA_session -> DAA_digest || 1 ||`
5466 `inputData1`)

5467 h. If `selector == 0`, verify that `inputData1` is a handle to a TPM identity key (AIK), and
5468 return error `TPM_DAA_INPUT_DATA1` on mismatch

5469 i. Set `DAA_session -> DAA_digest` to SHA-1 (`DAA_session -> DAA_digest || 0 || n2`)
5470 where `n2` is the modulus of the AIK

5471 j. Set `outputData = DAA_session -> DAA_digest`

5472 k. increment `DAA_session -> DAA_stage` by 1

5473 l. return `TPM_SUCCESS`.

5474 11.If `stage==11`

5475 a. Verify that `DAA_session ->DAA_stage==11`. Return `TPM_DAA_STAGE` and flush
5476 handle on mismatch

5477 b. Verify that `DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and
5478 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5479 c. Verify that `DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific)` and
5480 return error `TPM_DAA_TPM_SETTINGS` on mismatch

5481 d. Obtain DAA_SIZE_r0 bytes using the MGF1 function and label them r0. “r0” ||
5482 DAA_session -> DAA_contextSeed is the Z seed.

5483 e. Set $f = \text{SHA-1}(\text{DAA_tpmSpecific} \rightarrow \text{DAA_rekey} || \text{DAA_tpmSpecific} \rightarrow \text{DAA_count} ||$
5484 $0) || \text{SHA-1}(\text{DAA_tpmSpecific} \rightarrow \text{DAA_rekey} || \text{DAA_tpmSpecific} \rightarrow \text{DAA_count} || 1)$
5485 $\text{mod DAA_issuerSettings} \rightarrow \text{DAA_generic_q}$.

5486 f. Set $f_0 = f \text{ mod } 2^{\text{DAA_power0}}$ (erase all but the lowest DAA_power0 bits of f)

5487 g. Set $s_0 = r_0 + (\text{DAA_session} \rightarrow \text{DAA_digest}) * (f_0)$

5488 h. set outputData = s0

5489 i. increment DAA_session -> DAA_stage by 1

5490 j. return TPM_SUCCESS

5491 12.If stage==12

5492 a. Verify that DAA_session ->DAA_stage==12. Return TPM_DAA_STAGE and flush
5493 handle on mismatch

5494 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5495 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5496 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and
5497 return error TPM_DAA_TPM_SETTINGS on mismatch

5498 d. Obtain DAA_SIZE_r1 bytes using the MGF1 function and label them r1. “r1” ||
5499 DAA_session -> DAA_contextSeed is the Z seed.

5500 e. Set $f = \text{SHA-1}(\text{DAA_tpmSpecific} \rightarrow \text{DAA_rekey} || \text{DAA_tpmSpecific} \rightarrow \text{DAA_count} ||$
5501 $0) || \text{SHA-1}(\text{DAA_tpmSpecific} \rightarrow \text{DAA_rekey} || \text{DAA_tpmSpecific} \rightarrow \text{DAA_count} || 1)$
5502 $\text{mod DAA_issuerSettings} \rightarrow \text{DAA_generic_q}$.

5503 f. Shift f right by DAA_power0 bits (discard the lowest DAA_power0 bits) and label the
5504 result f1

5505 g. Set $s_1 = r_1 + (\text{DAA_session} \rightarrow \text{DAA_digest}) * (f_1)$

5506 h. set outputData = s1

5507 i. increment DAA_session -> DAA_stage by 1

5508 j. return TPM_SUCCESS

5509 13.If stage==13

5510 a. Verify that DAA_session ->DAA_stage==13. Return TPM_DAA_STAGE and flush
5511 handle on mismatch

5512 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5513 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5514 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and
5515 return error TPM_DAA_TPM_SETTINGS on mismatch

5516 d. Set DAA_private_v0= unwrap(inputData0) using TPM_PERMANENT_DATA ->
5517 daaBlobKey

5518 e. Verify that $\text{SHA-1}(\text{DAA_private_v0}) == \text{DAA_tpmSpecific} \rightarrow \text{DAA_digest_v0}$ and return
5519 error TPM_DAA_INPUT_DATA0 on mismatch

5520 f. Obtain DAA_SIZE_r2 bytes from the MGF1 function and label them r2. “r2” ||
5521 DAA_session -> DAA_contextSeed is the Z seed.

5522 g. Set $s2 = r2 + (DAA_session \rightarrow DAA_digest) * (DAA_private_v0) \bmod 2^{DAA_power1}$
5523 (erase all but the lowest DAA_power1 bits of s2)

5524 h. set outputData = s2

5525 i. increment DAA_session -> DAA_stage by 1

5526 j. return TPM_SUCCESS

5527 14.If stage==14

5528 a. Verify that DAA_session ->DAA_stage==14. Return TPM_DAA_STAGE and flush
5529 handle on mismatch

5530 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5531 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5532 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and
5533 return error TPM_DAA_TPM_SETTINGS on mismatch

5534 d. Set DAA_private_v0= unwrap(inputData0) using TPM_PERMANENT_DATA ->
5535 daaBlobKey

5536 e. Verify that SHA-1(DAA_private_v0) == DAA_tpmSpecific -> DAA_digest_v0 and return
5537 error TPM_DAA_INPUT_DATA0 on mismatch

5538 f. Obtain DAA_SIZE_r2 bytes using the MGF1 function and label them r2. “r2” ||
5539 DAA_session -> DAA_contextSeed is the Z seed.

5540 g. Set $s12 = r2 + (DAA_session \rightarrow DAA_digest) * (DAA_private_v0)$.

5541 h. Shift s12 right by DAA_power1 bits (erase the lowest DAA_power1 bits).

5542 i. Set DAA_session -> DAA_scratch = s12

5543 j. set outputData = NULL

5544 k. increment DAA_session -> DAA_stage by 1

5545 l. return TPM_SUCCESS

5546 15.If stage==15

5547 a. Verify that DAA_session ->DAA_stage==15. Return TPM_DAA_STAGE and flush
5548 handle on mismatch

5549 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5550 return error TPM_DAA_ISSUER_SETTINGS on mismatch

5551 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific) and
5552 return error TPM_DAA_TPM_SETTINGS on mismatch

5553 d. Set DAA_private_v1 = unwrap(inputData0) using TPM_PERMANENT_DATA ->
5554 daaBlobKey

5555 e. Verify that SHA-1(DAA_private_v1) == DAA_tpmSpecific -> DAA_digest_v1 and return
5556 error TPM_DAA_INPUT_DATA0 on mismatch

5557 f. Obtain DAA_SIZE_r4 bytes using the MGF1 function and label them r4. “r4” ||
5558 DAA_session -> DAA_contextSeed is the Z seed.

5559 g. Set $s3 = r4 + (DAA_session \rightarrow DAA_digest) * (DAA_private_v1) + (DAA_session \rightarrow$
5560 $DAA_scratch)$.

5561 h. Set $DAA_session \rightarrow DAA_scratch = NULL$

5562 i. set $outputData = s3$

5563 j. Terminate the DAA session and all resources associated with the DAA sign session
5564 handle.

5565 k. return $TPM_SUCCESS$

5566 16.If $stage > 15$, return error: TPM_DAA_STAGE

5567 **27. Deprecated commands**

5568 **Start of informative comment:**

5569 This section covers the commands that were in version 1.1 but now have new functionality
5570 in other functions. The deprecated commands are still available in 1.2 but all new software
5571 should use the new functionality.

5572 There is no requirement that the deprecated commands work with new structures.

5573 **End of informative comment.**

- 5574 1. Commands deprecated in version 1.2 **MUST** work with version 1.1 structures
5575 2. Commands deprecated in version 1.2 **MAY** work with version 1.2 structures

5576 **27.1 Key commands**5577 **Start of informative comment:**

5578 The key commands are deprecated as the new way to handle keys is to use the standard
5579 context commands. So TPM_EvictKey is now handled by TPM_FlushSpecific,
5580 TPM_Terminate_Handle by TPM_FlushSpecific.

5581 **End of informative comment.**5582 **27.1.1 TPM_EvictKey**5583 **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.

5584 **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

5585 **Actions**

5586 The TPM will invalidate the key stored in the specified handle and return the space to the
5587 available internal pool for subsequent query by TPM_GetCapability and usage by
5588 TPM_LoadKey. If the specified key handle does not correspond to a valid key, an error will
5589 be returned.

5590 **New 1.2 functionality**

5591 The command must check the status of the ownerEvict flag for the key and if the flag is
5592 TRUE return TPM_KEY_CONTROL_OWNER

5593 **27.1.2 TPM_Terminate_Handle**

5594 **Start of informative comment:**

5595 This allows the TPM manager to clear out information in a session handle.

5596 The TPM may maintain the authorization session even though a key attached to it has been
5597 unloaded or the authorization session itself has been unloaded in some way. When a
5598 command is executed that requires this session, it is the responsibility of the external
5599 software to load both the entity and the authorization session information prior to
5600 command execution.

5601 **End of informative comment.**

5602 **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

5603 **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.

5604 **Descriptions**

5605 The TPM SHALL terminate the session and destroy all data associated with the session
5606 indicated.

5607 **Actions**

5608 A TPM SHALL unilaterally perform the actions of TPM_Terminate_Handle upon detection of
5609 the following events:

- 5610 1. Completion of a received command whose authorization “continueUse” flag is FALSE.
5611 2. Completion of a received command when a shared secret derived from the authorization
5612 session was exclusive-or’ed with data (to provide confidentiality for that data). This
5613 occurs during execution of a TPM_ChangeAuth command, for example.
5614 3. When the associated entity is destroyed (in the case of TPM Owner or SRK, for example)
5615 4. Upon execution of TPM_Init

- 5616 5. When the command returns an error. This is due to the fact that when returning an
5617 error the TPM does not send back nonceEven. There is no way to maintain the rolling
5618 nonces, hence the TPM MUST terminate the authorization session.
- 5619 6. Failure of an authorization check belonging to that authorization session.

5620 **27.2 Context management**

5621 **Start of informative comment:**

5622 The 1.1 context commands were written for specific resource types. The 1.2 commands are
5623 generic for all resource types. So the Savexxx commands are replaced by TPM_SaveContext
5624 and the LoadXXX commands by TPM_LoadContext.

5625 **End of informative comment.**

5626 **27.2.1 TPM_SaveKeyContext**

5627 **Start of informative comment:**

5628 TPM_SaveKeyContext saves a loaded key outside the TPM. After creation of the key context
5629 blob the TPM automatically releases the internal memory used by that key. The format of
5630 the key context blob is specific to a TPM.

5631 **End of informative comment.**

5632 **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

5633 **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.

5634 **Description**

- 5635 1. This command allows saving a loaded key outside the TPM. After creation of the
5636 keyContextBlob, the TPM automatically releases the internal memory used by that key.
5637 The format of the key context blob is specific to a TPM.
- 5638 2. A TPM protected capability belonging to the TPM that created a key context blob MUST
5639 be the only entity that can interpret the contents of that blob. If a cryptographic
5640 technique is used for this purpose, the level of security provided by that technique
5641 SHALL be at least as secure as a 2048 bit RSA algorithm. Any secrets (such as keys)

- 5642 used in such a cryptographic technique MUST be generated using the TPM's random
5643 number generator. Any symmetric key MUST be used within the power-on session
5644 during which it was created, only.
- 5645 3. A key context blob SHALL enable verification of the integrity of the contents of the blob
5646 by a TPM protected capability.
- 5647 4. A key context blob SHALL enable verification of the session validity of the contents of the
5648 blob by a TPM protected capability. The method SHALL ensure that all key context blobs
5649 are rendered invalid if power to the TPM is interrupted.

5650 **27.2.2 TPM_LoadKeyContext**

5651 **Start of informative comment:**

5652 TPM_LoadKeyContext loads a key context blob into the TPM previously retrieved by a
5653 TPM_SaveKeyContext call. After successful completion the handle returned by this
5654 command can be used to access the key.

5655 **End of informative comment.**

5656 **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.

5657 **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.

5658 **Description**

- 5659 1. This command allows loading a key context blob into the TPM previously retrieved by a
5660 TPM_SaveKeyContext call. After successful completion the handle returned by this
5661 command can be used to access the key.
- 5662 2. The contents of a key context blob SHALL be discarded unless the contents have passed
5663 an integrity test. This test SHALL (statistically) prove that the contents of the blob are
5664 the same as when the blob was created.
- 5665 3. The contents of a key context blob SHALL be discarded unless the contents have passed
5666 a session validity test. This test SHALL (statistically) prove that the blob was created by
5667 this TPM during this power-on session.

5668 **27.2.3 TPM_SaveAuthContext**5669 **Start of informative comment:**

5670 TPM_SaveAuthContext saves a loaded authorization session outside the TPM. After creation
5671 of the authorization context blob, the TPM automatically releases the internal memory used
5672 by that session. The format of the authorization context blob is specific to a TPM.

5673 **End of informative comment.**5674 **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

5675 **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.

5676 **Description**

5677 This command allows saving a loaded authorization session outside the TPM. After creation
5678 of the authContextBlob, the TPM automatically releases the internal memory used by that
5679 session. The format of the authorization context blob is specific to a TPM.

5680 A TPM protected capability belonging to the TPM that created an authorization context blob
5681 MUST be the only entity that can interpret the contents of that blob. If a cryptographic
5682 technique is used for this purpose, the level of security provided by that technique SHALL
5683 be at least as secure as a 2048 bit RSA algorithm. Any secrets (such as keys) used in such a
5684 cryptographic technique MUST be generated using the TPM's random number generator.
5685 Any symmetric key MUST be used within the power-on session during which it was created,
5686 only.

5687 An authorization context blob SHALL enable verification of the integrity of the contents of
5688 the blob by a TPM protected capability.

5689 An authorization context blob SHALL enable verification of the session validity of the
5690 contents of the blob by a TPM protected capability. The method SHALL ensure that all
5691 authorization context blobs are rendered invalid if power to the TPM is interrupted.

5692 **27.2.4 TPM_LoadAuthContext**

5693 **Start of informative comment:**

5694 TPM_LoadAuthContext loads an authorization context blob into the TPM previously
5695 retrieved by a TPM_SaveAuthContext call. After successful completion the handle returned
5696 by this command can be used to access the authorization session.

5697 **End of informative comment.**

5698 **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.

5699 **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.

5700 **Description**

5701 This command allows loading an authorization context blob into the TPM previously
5702 retrieved by a TPM_SaveAuthContext call. After successful completion the handle returned
5703 by this command can be used to access the authorization session.

5704 The contents of an authorization context blob SHALL be discarded unless the contents have
5705 passed an integrity test. This test SHALL (statistically) prove that the contents of the blob
5706 are the same as when the blob was created.

5707 The contents of an authorization context blob SHALL be discarded unless the contents have
5708 passed a session validity test. This test SHALL (statistically) prove that the blob was created
5709 by this TPM during this power-on session.

5710 **27.3 DIR commands**5711 **Start of informative comment:**

5712 The DIR commands are replaced by the NV storage commands.

5713 The DIR [0] in 1.1 is now TPM_PERMANENT_DATA -> authDIR[0] and is always available for
5714 the TPM to use. It is accessed by DIR commands using dirIndex 0 and by NV commands
5715 using nvIndex TPM_NV_INDEX_DIR.

5716 If the TPM vendor supports additional DIR registers, the TPM vendor may return errors or
5717 provide vendor specific mappings for those DIR registers to NV storage locations.

5718 **End of informative comment.**

5719 1. A dirIndex value of 0 MUST corresponds to an NV storage nvIndex value
5720 TPM_NV_INDEX_DIR.

5721 2. The TPM vendor MAY return errors or MAY provide vendor specific mappings for DIR
5722 dirIndex values greater than 0 to NV storage locations.

5723 **27.3.1 TPM_DirWriteAuth**

5724 **Start of informative comment:**

5725 The TPM_DirWriteAuth operation provides write access to the Data Integrity Registers. DIRs
5726 are non-volatile memory registers held in a TPM-shielded location. Owner authentication is
5727 required to authorize this action.

5728 Access is also provided through the NV commands with nvIndex TPM_NV_INDEX_DIR.
5729 Owner authorization is not required when nvLocked is FALSE.

5730 Version 1.2 requires only one DIR. If the DIR named does not exist, the TPM_DirWriteAuth
5731 operation returns TPM_BADINDEX.

5732 **End of informative comment.**

5733 **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.

5734 **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.

5735 **Actions**

- 5736 1. Validate that authHandle contains a TPM Owner AuthData to execute the
- 5737 TPM_DirWriteAuth command
- 5738 2. Validate that dirIndex points to a valid DIR on this TPM
- 5739 3. Write newContents into the DIR pointed to by dirIndex

5740

5741 **27.3.2 TPM_DirRead**

5742 **Start of informative comment:**

5743 The TPM_DirRead operation provides read access to the DIRs. No authentication is required
5744 to perform this action because typically no cryptographically useful AuthData is available
5745 early in boot. TSS implementers may choose to provide other means of authorizing this
5746 action. Version 1.2 requires only one DIR. If the DIR named does not exist, the
5747 TPM_DirRead operation returns TPM_BADINDEX.

5748 **End of informative comment.**

5749 **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

5750 **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

5751 **Actions**

- 5752 1. Validate that dirIndex points to a valid DIR on this TPM
- 5753 2. Return the contents of the DIR in dirContents

5754 **27.4 Change Auth**

5755 **Start of informative comment:**

5756 The change auth commands can be duplicated by creating a transport session with
5757 confidentiality and issuing the changeAuth command.

5758 **End of informative comment.**

5759 **27.4.1 TPM_ChangeAuthAsymStart**

5760 **Start of informative comment:**

5761 The TPM_ChangeAuthAsymStart starts the process of changing AuthData for an entity. It
5762 sets up an OIAP session that must be retained for use by its twin
5763 TPM_ChangeAuthAsymFinish command.

5764 TPM_ChangeAuthAsymStart creates a temporary asymmetric public key “tempkey” to
5765 provide confidentiality for new AuthData to be sent to the TPM. TPM_ChangeAuthAsymStart
5766 certifies that tempkey was generated by a genuine TPM, by generating a certifyInfo
5767 structure that is signed by a TPM identity. The owner of that TPM identity must cooperate
5768 to produce this command, because TPM_ChangeAuthAsymStart requires authorization to
5769 use that identity.

5770 It is envisaged that tempkey and certifyInfo are given to the owner of the entity whose
5771 authorization is to be changed. That owner uses certifyInfo and a
5772 TPM_IDENTITY_CREDENTIAL to verify that tempkey was generated by a genuine TPM. This
5773 is done by verifying the TPM_IDENTITY_CREDENTIAL using the public key of a CA,
5774 verifying the signature on the certifyInfo structure with the public key of the identity in
5775 TPM_IDENTITY_CREDENTIAL, and verifying tempkey by comparing its digest with the value
5776 inside certifyInfo. The owner uses tempkey to encrypt the desired new AuthData and inserts
5777 that encrypted data in a TPM_ChangeAuthAsymFinish command, in the knowledge that
5778 only a TPM with a specific identity can interpret the new AuthData.

5779 **End of informative comment.**

5780 **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.

5781 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuthAsymStart
7	95	3S	95	TPM_CERTIFY_INFO	certifyInfo	The certifyInfo structure that is to be signed.
8	4	4S	4	UINT32	sigSize	The used size of the output area for the signature
9	<>	5S	<>	BYTE[]	sig	The signature of the certifyInfo parameter.
10	4	6s	4	TPM_KEY_HANDLE	ephHandle	The keyHandle identifier to be used by ChangeAuthAsymFinish for the ephemeral key
11	<>	7S	<>	TPM_KEY	tempKey	Structure containing all parameters and public part of ephemeral key. TPM_KEY.encSize is set to 0.
12	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
13	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
14	20			TPM_AUTHDATA	resAuth	Authorization. HMAC key: idKey.usageAuth.

5782 **Actions**

- 5783 1. The TPM SHALL verify the AuthData to use the TPM identity key held in idHandle. The
5784 TPM MUST verify that the key is a TPM identity key.
- 5785 2. The TPM SHALL validate the algorithm parameters for the key to create from the
5786 tempKey parameter.
- 5787 3. Recommended key type is RSA
- 5788 4. Minimum RSA key size MUST is 512 bits, recommended RSA key size is 1024
- 5789 5. For other key types the minimum key size strength MUST be comparable to RSA 512
- 5790 6. If the TPM is not designed to create a key of the requested type, return the error code
5791 TPM_BAD_KEY_PROPERTY
- 5792 7. The TPM SHALL create a new key (k1) in accordance with the algorithm parameter. The
5793 newly created key is pointed to by ephHandle.
- 5794 8. The TPM SHALL fill in all fields in tempKey using k1 for the information. The TPM_KEY -
5795 > encSize MUST be 0.
- 5796 9. The TPM SHALL fill in certifyInfo using k1 for the information. The certifyInfo -> data
5797 field is supplied by the antiReplay.
- 5798 10. The TPM then signs the certifyInfo parameter using the key pointed to by idHandle. The
5799 resulting signed blob is returned in sig parameter

5800

Field 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_PARAMS	info	This specifies the type of key and its parameters.
BOOL	parentPCRStatus	This SHALL be set to FALSE.

5801

5802 **27.4.2 TPM_ChangeAuthAsymFinish**5803 **Start of informative comment:**

5804 The TPM_ChangeAuth command allows the owner of an entity to change the AuthData for
5805 the entity.

5806 The command requires the cooperation of the owner of the parent of the entity, since
5807 AuthData must be provided to use that parent entity. The command requires knowledge of
5808 the existing AuthData information and passes the new AuthData information. The
5809 newAuthLink parameter proves knowledge of existing AuthData information and new
5810 AuthData information. The new AuthData information “encNewAuth” is encrypted using the
5811 “tempKey” variable obtained via TPM_ChangeAuthAsymStart.

5812 A parent therefore retains control over a change in the AuthData of a child, but is prevented
5813 from knowing the new AuthData for that child.

5814 The changeProof parameter provides a proof that the new AuthData value was properly
5815 inserted into the entity. The inclusion of a nonce from the TPM provides an entropy source
5816 in the case where the AuthData value may be in itself be a low entropy value (hash of a
5817 password etc).

5818 **End of informative comment.**5819 **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.

5820

5821 **Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuthAsymFinish
4	4	3S	4	UINT32	outDataSize	The used size of the output area for outData
5	<>	4S	<>	BYTE[]	outData	The modified, encrypted entity.
6	20	5s	20	TPM_NONCE	saltNonce	A nonce value from the TPM RNG to add entropy to the changeProof value
7	<>	6S	<>	TPM_DIGEST	changeProof	Proof that 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.

5822 **Description**

5823 If the parentHandle points to the SRK then the HMAC key MUST be built using the TPM
5824 Owner authentication.

5825 **Actions**

- 5826 1. The TPM SHALL validate that the authHandle parameter authorizes use of the key in
5827 parentHandle.
- 5828 2. The encData field MUST be the encData field from TPM_STORED_DATA or TPM_KEY.
- 5829 3. The TPM SHALL create e1 by decrypting the entity held in the encData parameter.
- 5830 4. The TPM SHALL create a1 by decrypting encNewAuth using the ephHandle ->
5831 TPM_KEY_AUTHCHANGE private key. a1 is a structure of type
5832 TPM_CHANGEAUTH_VALIDATE.
- 5833 5. The TPM SHALL create b1 by performing the following HMAC calculation: b1 = HMAC
5834 (a1 -> newAuthSecret). The secret for this calculation is encData -> currentAuth. This
5835 means that b1 is a value built from the current AuthData value (encData ->
5836 currentAuth) and the new AuthData value (a1 -> newAuthSecret).
- 5837 6. The TPM SHALL compare b1 with newAuthLink. The TPM SHALL indicate a failure if the
5838 values do not match.
- 5839 7. The TPM SHALL replace e1 -> authData with a1 -> newAuthSecret
- 5840 8. The TPM SHALL encrypt e1 using the appropriate functions for the entity type. The key
5841 to encrypt with is parentHandle.
- 5842 9. The TPM SHALL create saltNonce by taking the next 20 bytes from the TPM RNG.

- 5843 10.The TPM SHALL create changeProof a HMAC of (saltNonce concatenated with a1 -> n1)
5844 using a1 -> newAuthSecret as the HMAC secret.
- 5845 11.The TPM MUST destroy the TPM_KEY_AUTHCHANGE key associated with the
5846 authorization session.

5847 27.5 TPM_Reset

5848 Start of informative comment:

5849 TPM_Reset releases all resources associated with existing authorization sessions. This is
5850 useful if a TSS driver has lost track of the state in the TPM.

5851 End of informative comment.

5852 Deprecated Command in 1.2

5853 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.

5854 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.

5855 Description

5856 This is a deprecated command in V1.2. This command in 1.1 only referenced authorization
5857 sessions and is not upgraded to affect any other TPM entity in 1.2

5858 Actions

- 5859 1. The TPM invalidates all resources allocated to authorization sessions as per version 1.1
5860 extant in the TPM
- 5861 a. This includes structures created by TPM_SaveAuthContext and TPM_SaveKeyContext
5862 b. Structures created by TPM_Contextxxx (the new 1.2 commands) are not affected by
5863 this command
- 5864 2. The TPM does not reset any PCR or DIR values.
- 5865 3. The TPM does not reset any flags in the TPM_STCLEAR_FLAGS structure.
- 5866 4. The TPM does not reset or invalidate any keys

5867 **27.6 TPM_OwnerReadPubek**5868 **Start of informative comment:**

5869 Return the endorsement key public portion. This is authorized by the TPM Owner.

5870 **End of informative comment.**5871 **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.

5872 **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.

5873 **Description**

5874 This command returns the PUBEK.

5875 **Actions**

5876 The TPM_OwnerReadPubek command SHALL

- 5877 1. Validate the TPM Owner AuthData to execute this command
- 5878 2. Export the PUBEK

5879 **27.7 TPM_DisablePubekRead**

5880 **Start of informative comment:**

5881 The TPM Owner may wish to prevent any entity from reading the PUBEK. This command
5882 sets the non-volatile flag so that the TPM_ReadPubek command always returns
5883 TPM_DISABLED_CMD.

5884 This command has in essence been deprecated as TPM_TakeOwnership now sets the value
5885 to false. The command remains at this time for backward compatibility.

5886 **End of informative comment.**

5887 **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.

5888 **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.

5889 **Actions**

5890 1. This capability sets the TPM_PERMANENT_FLAGS -> readPubek flag to FALSE.

5891 **27.8 TPM_LoadKey**5892 **Start of informative comment:**

5893 Version 1.2 deprecates TPM_LoadKey due to the HMAC of the new key handle on return.
5894 The wrapping makes use of the handle difficult in an environment where the TSS, or other
5895 management entity, is changing the TPM handle to a virtual handle.

5896 Software using TPM_LoadKey on a 1.2 TPM can have a collision with the returned handle as
5897 the 1.2 TPM uses random values in the lower three bytes of the handle. All new software
5898 must use LoadKey2 to allow management software the ability to manage the key handle.

5899 Before the TPM can use a key to either wrap, unwrap, bind, unbind, seal, unseal, sign or
5900 perform any other action, it needs to be present in the TPM. The TPM_LoadKey function
5901 loads the key into the TPM for further use.

5902 The TPM assigns the key handle. The TPM always locates a loaded key by use of the handle.
5903 The assumption is that the handle may change due to key management operations. It is the
5904 responsibility of upper level software to maintain the mapping between handle and any
5905 label used by external software.

5906 This command has the responsibility of enforcing restrictions on the use of keys. For
5907 example, when attempting to load a STORAGE key it will be checked for the restrictions on
5908 a storage key (2048 size etc.).

5909 The load command must maintain a record of whether any previous key in the key
5910 hierarchy was bound to a PCR using parentPCRStatus.

5911 The flag parentPCRStatus enables the possibility of checking that a platform passed
5912 through some particular state or states before finishing in the current state. A grandparent
5913 key could be linked to state-1, a parent key could be linked to state-2, and a child key could be
5914 linked to state-3, for example. The use of the child key then indicates that the platform
5915 passed through states 1 and 2 and is currently in state 3, in this example. TPM_Startup
5916 with stType == TPM_ST_CLEAR indicates that the platform has been reset, so the platform
5917 has not passed through the previous states. Hence keys with parentPCRStatus==TRUE
5918 must be unloaded if TPM_Startup is issued with stType == TPM_ST_CLEAR.

5919 If a TPM_KEY structure has been decrypted AND the integrity test using "pubDataDigest"
5920 has passed AND the key is non-migratory, the key must have been created by the TPM. So
5921 there is every reason to believe that the key poses no security threat to the TPM. While there
5922 is no known attack from a rogue migratory key, there is a desire to verify that a loaded
5923 migratory key is a real key, arising from a general sense of unease about execution of
5924 arbitrary data as a key. Ideally a consistency check would consist of an encrypt/decrypt
5925 cycle, but this may be expensive. For RSA keys, it is therefore suggested that the
5926 consistency test consists of dividing the supposed RSA product by the supposed RSA prime,
5927 and checking that there is no remainder.

5928 **End of informative comment.**

5929 **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_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.

5930 **Outgoing Operands and Sizes**

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

5931 **Actions**

5932 The TPM SHALL perform the following steps:

- 5933 1. Validate the command and the parameters using parentAuth and parentHandle ->
5934 usageAuth
- 5935 2. If parentHandle -> keyUsage is NOT TPM_KEY_STORAGE return
5936 TPM_INVALID_KEYUSAGE
- 5937 3. If the TPM is not designed to operate on a key of the type specified by inKey, return the
5938 error code TPM_BAD_KEY_PROPERTY
- 5939 4. The TPM MUST handle both TPM_KEY and TPM_KEY12 structures
- 5940 5. Decrypt the inKey -> privkey to obtain TPM_STORE_ASYMKEY structure using the key
5941 in parentHandle

- 5942 6. Validate the integrity of inKey and decrypted TPM_STORE_ASYMKEY
- 5943 a. Reproduce inKey -> TPM_STORE_ASYMKEY -> pubDataDigest using the fields of
- 5944 inKey, and check that the reproduced value is the same as pubDataDigest
- 5945 7. Validate the consistency of the key and it's key usage.
- 5946 a. If inKey -> keyFlags -> migratable is TRUE, the TPM SHALL verify consistency of the
- 5947 public and private components of the asymmetric key pair. If inKey -> keyFlags ->
- 5948 migratable is FALSE, the TPM MAY verify consistency of the public and private
- 5949 components of the asymmetric key pair. The consistency of an RSA key pair MAY be
- 5950 verified by dividing the supposed (P*Q) product by a supposed prime and checking that
- 5951 there is no remainder.
- 5952 b. If inKey -> keyUsage is TPM_KEY_IDENTITY, verify that inKey->keyFlags->migratable
- 5953 is FALSE. If it is not, return TPM_INVALID_KEYUSAGE
- 5954 c. If inKey -> keyUsage is TPM_KEY_AUTHCHANGE, return TPM_INVALID_KEYUSAGE
- 5955 d. If inKey -> keyFlags -> migratable equals 0 then verify that TPM_STORE_ASYMKEY -
- 5956 > migrationAuth equals TPM_PERMANENT_DATA -> tpmProof
- 5957 e. Validate the mix of encryption and signature schemes
- 5958 f. If TPM_PERMANENT_FLAGS -> FIPS is TRUE then
- 5959 i. If keyInfo -> keySize is less than 1024 return TPM_NOTFIPS
- 5960 ii. If keyInfo -> authDataUsage specifies TPM_AUTH_NEVER return TPM_NOTFIPS
- 5961 iii. If keyInfo -> keyUsage specifies TPM_KEY_LEGACY return TPM_NOTFIPS
- 5962 g. If inKey -> keyUsage is TPM_KEY_STORAGE or TPM_KEY_MIGRATE
- 5963 i. algorithmID MUST be TPM_ALG_RSA
- 5964 ii. Key size MUST be 2048
- 5965 iii. sigScheme MUST be TPM_SS_NONE
- 5966 h. If inKey -> keyUsage is TPM_KEY_IDENTITY
- 5967 i. algorithmID MUST be TPM_ALG_RSA
- 5968 ii. Key size MUST be 2048
- 5969 iii. encScheme MUST be TPM_ES_NONE
- 5970 i. If the decrypted inKey -> pcrInfo is NULL,
- 5971 i. The TPM MUST set the internal indicator to indicate that the key is not using any
- 5972 PCR registers.
- 5973 j. Else
- 5974 i. The TPM MUST store pcrInfo in a manner that allows the TPM to calculate a
- 5975 composite hash whenever the key will be in use
- 5976 ii. The TPM MUST handle both version 1.1 TPM_PCR_INFO and 1.2
- 5977 TPM_PCR_INFO_LONG structures according to the type of TPM_KEY structure
- 5978 iii. The TPM MUST validate the TPM_PCR_INFO or TPM_PCR_INFO_LONG
- 5979 structures

- 5980 8. Perform any processing necessary to make TPM_STORE_ASYMKEY key available for
5981 operations
- 5982 9. Load key and key information into internal memory of the TPM. If insufficient memory
5983 exists return error TPM_NOSPACE.
- 5984 10. Assign inKeyHandle according to internal TPM rules.
- 5985 11. Set InKeyHandle -> parentPCRStatus to parentHandle -> parentPCRStatus.
- 5986 12. If ParentHandle indicates it is using PCR registers then set inKeyHandle ->
5987 parentPCRStatus to TRUE.

5988 **28. Deleted Commands**5989 **Start of informative comment:**

5990 These commands are no longer active commands. Their removal is due to security concerns
5991 with their use.

5992 **End of informative comment.**

- 5993 1. The TPM MUST return TPM_BAD_ORDINAL for any deleted command

5994 **28.1 TPM_GetCapabilitySigned**

5995 **Start of informative comment:**

5996 Along with TPM_GetCapabilityOwner this command allowed the possible signature of
5997 improper values.

5998 TPM_GetCapabilitySigned is almost the same as TPM_GetCapability. The differences are
5999 that the input includes a challenge (a nonce) and the response includes a digital signature
6000 to vouch for the source of the answer.

6001 If a caller itself requires proof, it is sufficient to use any signing key for which only the TPM
6002 and the caller have AuthData.

6003 If a caller requires proof for a third party, the signing key must be one whose signature is
6004 trusted by the third party. A TPM-identity key may be suitable.

6005 **End of informative comment.**

6006 **Deleted Ordinal**

6007 TPM_GetCapabilitySigned

6008 **28.2 TPM_GetOrdinalAuditStatus**6009 **Start of informative comment:**

6010 Get the status of the audit flag for the given ordinal.

6011 **End of informative comment.**6012 **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

6013 **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

6014 **Actions**

6015 1. The TPM returns the Boolean value for the given ordinal. The value is TRUE if the
6016 command is being audited.

6017 **28.3 TPM_CertifySelfTest**

6018 **Start of informative comment:**

6019 TPM_CertifySelfTest causes the TPM to perform a full self-test and return an authenticated
6020 value if the test passes.

6021 If a caller itself requires proof, it is sufficient to use any signing key for which only the TPM
6022 and the caller have AuthData.

6023 If a caller requires proof for a third party, the signing key must be one whose signature is
6024 trusted by the third party. A TPM-identity key may be suitable.

6025 **End of informative comment.**

6026 **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

6027 **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

6028 **Description**

6029 The key in keyHandle MUST have a KEYUSAGE value of type TPM_KEY_SIGNING or
6030 TPM_KEY_LEGACY or TPM_KEY_IDENTITY.

6031 Information returned by TPM_CertifySelfTest MUST NOT aid identification of an individual
6032 TPM.

6033 **Actions**

6034 1. The TPM SHALL perform TPM_SelfTestFull. If the test fails the TPM returns the
6035 appropriate error code.

6036 2. After successful completion of the self-test the TPM then validates the authorization to
6037 use the key pointed to by keyHandle

6038 a. If the key pointed to by keyHandle has a signature scheme that is not
6039 TPM_SS_RSASSAPKCS1v15_SHA1, the TPM may either return TPM_BAD_SCHEME or
6040 may return TPM_SUCCESS and a vendor specific signature.

6041 3. Create t1 the NOT null terminated string of "Test Passed", i.e. 11 bytes.

6042 4. The TPM creates m2 the message to sign by concatenating t1 || AntiReplay || ordinal.

6043 5. The TPM signs the SHA-1 of m2 using the key identified by keyHandle, and returns the
6044 signature as sig.

6045