

# TPM Main

## Part 3 Commands

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

Contact: [admin@trustedcomputinggroup.com](mailto:admin@trustedcomputinggroup.com)

## TCG Published

Copyright © 2003-2011 Trusted Computing Group, Incorporated

TCG

34Copyright © 2003-2009 Trusted Computing Group, Incorporated.

### 35**Disclaimers, Notices, and License Terms**

36THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER,  
37INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR  
38ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY  
39PROPOSAL, SPECIFICATION OR SAMPLE.

40Without limitation, TCG disclaims all liability, including liability for infringement of any  
41proprietary rights, relating to use of information in this specification and to the  
42implementation of this specification, and TCG disclaims all liability for cost of procurement  
43of substitute goods or services, lost profits, loss of use, loss of data or any incidental,  
44consequential, direct, indirect, or special damages, whether under contract, tort, warranty  
45or otherwise, arising in any way out of use or reliance upon this specification or any  
46information herein.

47This document is copyrighted by Trusted Computing Group (TCG), and no license, express or implied, is  
48granted herein other than as follows: You may not copy or reproduce the document or distribute it to others  
49without written permission from TCG, except that you may freely do so for the purposes of (a) examining or  
50implementing TCG specifications or (b) developing, testing, or promoting information technology standards and  
51best practices, so long as you distribute the document with these disclaimers, notices, and license terms.

52

53Contact the Trusted Computing Group at  
54<http://www.trustedcomputinggroup.org/> for information  
55on specification licensing through membership agreements.

56Any marks and brands contained herein are the property of their respective owners.

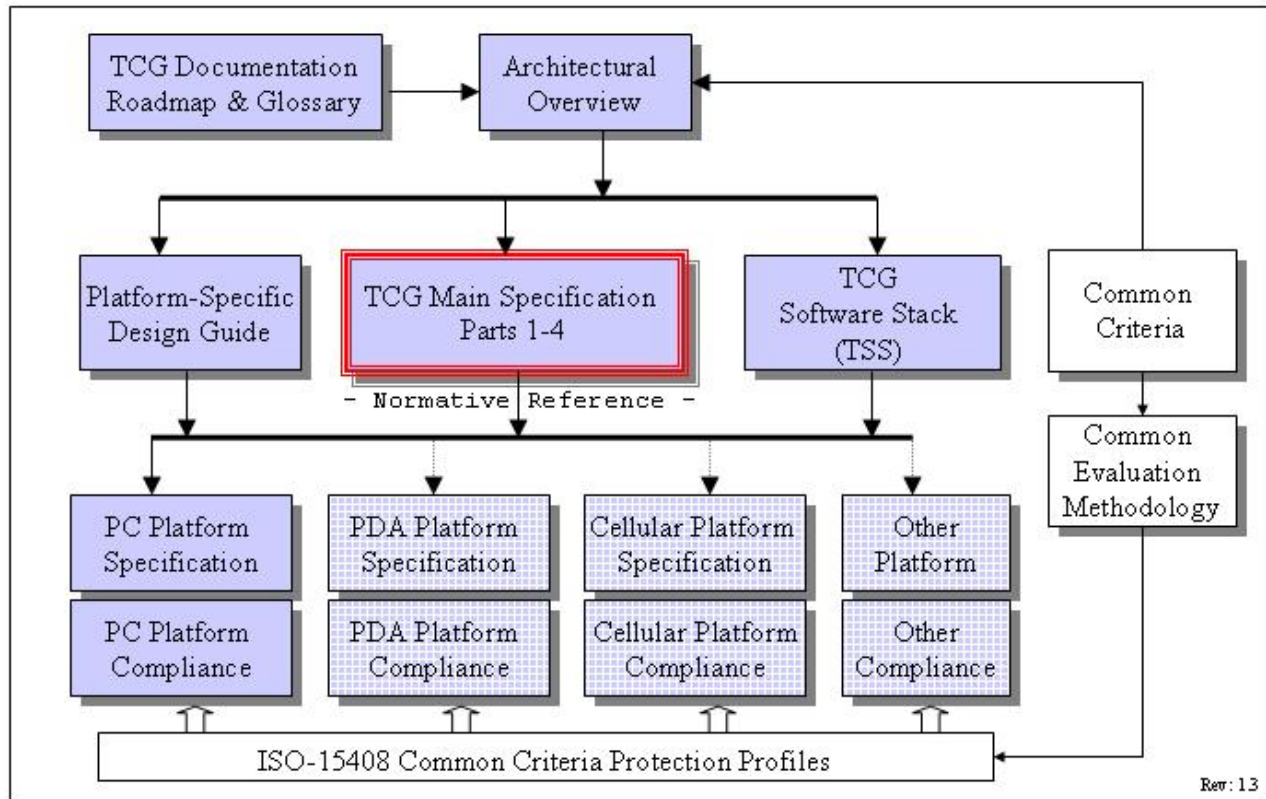
57

## 58Change History

Version	Date	Description
Rev 50	Jul 2003	Started 01 Jul 2003 by David Grawrock Breakup into parts and the merge of 1.1 commands
Rev 63	Oct 2003	Change history tied to part 1 and kept in part 1 (DP)
Rev 71	Mar 2004	Change in terms from authorization data to AuthData.
Rev 91	Sept 2005	The following modifications were made by Tasneem Brutch: <ul style="list-style-type: none"> <li>Update to section 6.2 informative, for TPM_OwnerClear.</li> <li>Addition of action item 15, to section 6.2, for TPM_OwnerClear.</li> <li>Addition of "MAY" to section 20.1, TPM_NV_DefineSpace, Action 1(a).</li> <li>Addition of a new Action (4) to Section 20.2, TPM_NV_WriteValue</li> <li>Addition of a new Action (3) to Section 20.4, TPM_NV_ReadValue.</li> <li>Typo corrected in Section 21.1</li> <li>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.</li> </ul>
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 disableFullDALogicInfo. Clarified that _INFO keys may be used where _SHA1 keys are used. Clarified that a global secret can be used for field upgrade confidentiality. Added TPM_CMK_CreateBlob actions for the migrationType parameter. Added TPM_CertifyKey action to check payload. Clarified that TPM_Delegate_LoadOwnerDelegation returns an error if there is no owner and owner authorization is present. Clarified that TPM_NV_DefineSpace cannot define the DIR index. Clarified that the TPM does not have to clean up the effects of a wrapped command upon failure of a transport response. Clarified that TPM_ReleaseCounter does not ignore the continueAuthSession parameter.
Rev 102	Sept 2006	Reworked TPM_GetPubkey to always check authorization data if present and allow no-authorization for TPM_AUTH_PRIV_USE_ONLY or TPM_AUTH_NEVER. Fixed TPM_LoadContext typo, Action 6.e. returns error if the HMAC does NOT match.
Rev 103	Oct 2006	Added warning notes where excluding key handle from HMAC can allow an attack. Added warning that delegating TPM_ChangeAuth allows elevation of privilege.
Rev 104	Nov 2006	Owner clear sets allowMaintenance and readSRKPub to default state. TPM_Unseal can use DSAP. TPM_CreateEndorsementKeyPair uses TPM_ES_RSAESOAEP_SHA1_MGF1.
Rev 105	Feb 2007	TPM_Seal, TPM_CreateWrapKey informative that they lack an identifier. TPM_NV_DefineSpace should check inputs before changing state. TPM_NV_DefineSpace, TPM_NV_WriteValue, TPM_NV_ReadValue ignore disabled and deactivated when nvLocked is FALSE, MAY always check HMAC. TPM_NV_WriteValue must not return error for DIR data size of 0. TPM_NV_ReadValue partial DIR reads are allowed. Informative that audit occurs twice for transport wrapped command. TPM_Reset must invalidate OSAP and DSAP sessions, must not invalidate sessions saved by TPM_SaveContext.
Rev 106	April 2007	Removed tpmProof check for non-migratable parent keys.
Rev 107	July 2007	Removed unused maxNVBufSize. Increment the auditMonotonicCounter before audit response if digest is zero. State should not change on field upgrade authorization failure. PCR values for a key are validated at use, not at load. TPM_StirRandom is not required to check for data < 256 bytes. TPM_ChangeAuth must validate usageAuth. Entity PCRs must be validated each time an OIAP session is used. TPM_ExecuteTransport MUST log public key logs.
Rev 108	Sept 2007	TPM_ForceClear succeeds even with no owner (informative). Audit only occurs when the commands executes successfully. Field upgrade should not change shielded locations. Reordered TPM_NV_DefineSpace, TPM_NV_WriteValue so the NV write count is not incremented if there is an authorization error.
Rev 109	Oct 2007	Added PCR index check to TPM_SHA1CompleteExtend, TPM_Extend, TPM_PCRRead.
Rev 110	May 2008	Minor typo corrections.

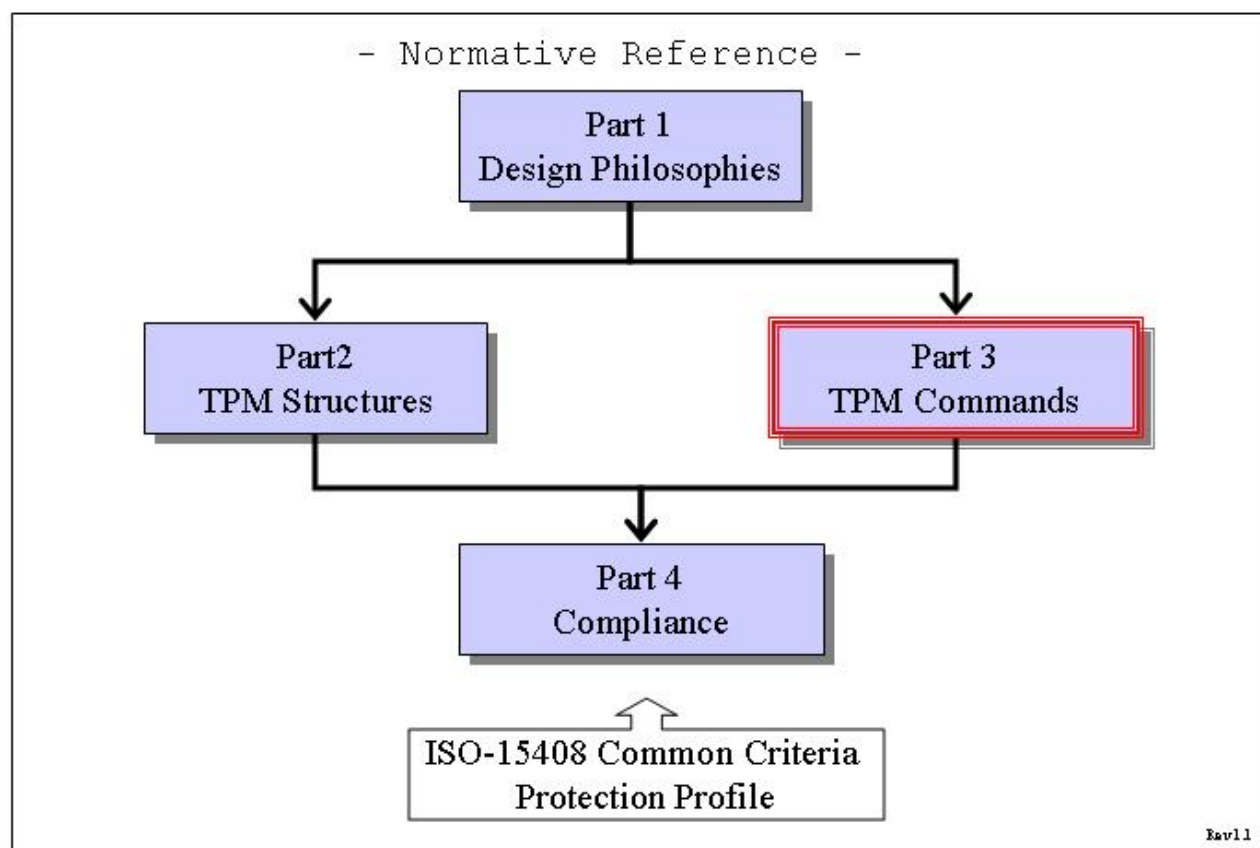
Rev 111	July 2008	TPM_SaveState gives priority to keys where parentPCRStatus is TRUE. Informative security warning when field upgrade adds new features. TPM_MakeIdentity normative that the signing key digestAtRelease is not validated.
Rev 112	Jan 2009	Create and load storage and migrate key, ownership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

# TCG Doc Roadmap – Main Spec



59

# TCG Main Spec Roadmap



60

## 61Table of Contents

62	<a href="#">1. Scope and Audience.....</a>	<a href="#">1</a>
63	<a href="#">1.1 Key words.....</a>	<a href="#">2</a>
64	<a href="#">1.2 Action Order.....</a>	<a href="#">3</a>
65	<a href="#">1.3 Statement Type.....</a>	<a href="#">4</a>
66	<a href="#">2. Description and TODO.....</a>	<a href="#">5</a>
67	<a href="#">3. Admin Startup and State.....</a>	<a href="#">6</a>
68	<a href="#">3.1 TPM_Init.....</a>	<a href="#">6</a>
69	<a href="#">3.2 TPM_Startup.....</a>	<a href="#">7</a>
70	<a href="#">3.3 TPM_SaveState.....</a>	<a href="#">10</a>
71	<a href="#">4. Admin Testing.....</a>	<a href="#">12</a>
72	<a href="#">4.1 TPM_SelfTestFull.....</a>	<a href="#">12</a>
73	<a href="#">4.2 TPM_ContinueSelfTest.....</a>	<a href="#">13</a>
74	<a href="#">4.3 TPM_GetTestResult.....</a>	<a href="#">15</a>
75	<a href="#">5. Admin Opt-in.....</a>	<a href="#">16</a>
76	<a href="#">5.1 TPM_SetOwnerInstall.....</a>	<a href="#">16</a>
77	<a href="#">5.2 TPM_OwnerSetDisable.....</a>	<a href="#">17</a>
78	<a href="#">5.3 TPM_PhysicalEnable.....</a>	<a href="#">18</a>
79	<a href="#">5.4 TPM_PhysicalDisable.....</a>	<a href="#">19</a>
80	<a href="#">5.5 TPM_PhysicalSetDeactivated.....</a>	<a href="#">20</a>
81	<a href="#">5.6 TPM_SetTempDeactivated.....</a>	<a href="#">21</a>
82	<a href="#">5.7 TPM_SetOperatorAuth.....</a>	<a href="#">23</a>
83	<a href="#">6. Admin Ownership.....</a>	<a href="#">24</a>
84	<a href="#">6.1 TPM_TakeOwnership.....</a>	<a href="#">24</a>
85	<a href="#">6.2 TPM_OwnerClear.....</a>	<a href="#">27</a>
86	<a href="#">6.3 TPM_ForceClear.....</a>	<a href="#">30</a>
87	<a href="#">6.4 TPM_DisableOwnerClear.....</a>	<a href="#">31</a>
88	<a href="#">6.5 TPM_DisableForceClear.....</a>	<a href="#">33</a>
89	<a href="#">6.6 TSC_PhysicalPresence.....</a>	<a href="#">34</a>
90	<a href="#">6.7 TSC_ResetEstablishmentBit.....</a>	<a href="#">37</a>
91	<a href="#">7. The Capability Commands.....</a>	<a href="#">38</a>
92	<a href="#">7.1 TPM_GetCapability.....</a>	<a href="#">39</a>
93	<a href="#">7.2 TPM_SetCapability.....</a>	<a href="#">41</a>
94	<a href="#">7.3 TPM_GetCapabilityOwner.....</a>	<a href="#">43</a>
95	<a href="#">8. Auditing.....</a>	<a href="#">45</a>
96	<a href="#">8.1 Audit Generation .....</a>	<a href="#">45</a>

32		
33		
34		
97	<a href="#">8.2 Effect of audit failing.....</a>	<a href="#">47</a>
98	<a href="#">8.3 TPM_GetAuditDigest.....</a>	<a href="#">48</a>
99	<a href="#">8.4 TPM_GetAuditDigestSigned.....</a>	<a href="#">50</a>
100	<a href="#">8.5 TPM_SetOrdinalAuditStatus.....</a>	<a href="#">53</a>
101	<a href="#">9. Administrative Functions - Management.....</a>	<a href="#">54</a>
102	<a href="#">9.1 TPM_FieldUpgrade.....</a>	<a href="#">54</a>
103	<a href="#">9.2 TPM_SetRedirection.....</a>	<a href="#">57</a>
104	<a href="#">9.3 TPM_ResetLockValue.....</a>	<a href="#">59</a>
105	<a href="#">10. Storage functions.....</a>	<a href="#">61</a>
106	<a href="#">10.1 TPM_Seal.....</a>	<a href="#">61</a>
107	<a href="#">10.2 TPM_Unseal.....</a>	<a href="#">65</a>
108	<a href="#">10.3 TPM_UnBind.....</a>	<a href="#">69</a>
109	<a href="#">10.4 TPM_CreateWrapKey.....</a>	<a href="#">72</a>
110	<a href="#">10.5 TPM_LoadKey2.....</a>	<a href="#">76</a>
111	<a href="#">10.6 TPM_GetPubKey.....</a>	<a href="#">80</a>
112	<a href="#">10.7 TPM_Sealx.....</a>	<a href="#">82</a>
113	<a href="#">11. Migration.....</a>	<a href="#">85</a>
114	<a href="#">11.1 TPM_CreateMigrationBlob.....</a>	<a href="#">85</a>
115	<a href="#">11.2 TPM_ConvertMigrationBlob.....</a>	<a href="#">89</a>
116	<a href="#">11.3 TPM_AuthorizeMigrationKey.....</a>	<a href="#">91</a>
117	<a href="#">11.4 TPM_MigrateKey.....</a>	<a href="#">93</a>
118	<a href="#">11.5 TPM_CMK_SetRestrictions.....</a>	<a href="#">95</a>
119	<a href="#">11.6 TPM_CMK_ApproveMA.....</a>	<a href="#">97</a>
120	<a href="#">11.7 TPM_CMK_CreateKey.....</a>	<a href="#">99</a>
121	<a href="#">11.8 TPM_CMK_CreateTicket.....</a>	<a href="#">102</a>
122	<a href="#">11.9 TPM_CMK_CreateBlob.....</a>	<a href="#">104</a>
123	<a href="#">11.10 TPM_CMK_ConvertMigration.....</a>	<a href="#">109</a>
124	<a href="#">12. Maintenance Functions (optional).....</a>	<a href="#">112</a>
125	<a href="#">12.1 TPM_CreateMaintenanceArchive.....</a>	<a href="#">114</a>
126	<a href="#">12.2 TPM_LoadMaintenanceArchive.....</a>	<a href="#">116</a>
127	<a href="#">12.3 TPM_KillMaintenanceFeature.....</a>	<a href="#">119</a>
128	<a href="#">12.4 TPM_LoadManuMaintPub.....</a>	<a href="#">120</a>
129	<a href="#">12.5 TPM_ReadManuMaintPub.....</a>	<a href="#">122</a>
130	<a href="#">13. Cryptographic Functions.....</a>	<a href="#">123</a>
131	<a href="#">13.1 TPM_SHA1Start.....</a>	<a href="#">123</a>
132	<a href="#">13.2 TPM_SHA1Update.....</a>	<a href="#">125</a>
133	<a href="#">13.3 TPM_SHA1Complete.....</a>	<a href="#">126</a>



134	<a href="#">13.4 TPM_SHA1CompleteExtend.....</a>	<a href="#">127</a>
135	<a href="#">13.5 TPM_Sign.....</a>	<a href="#">129</a>
136	<a href="#">13.6 TPM_GetRandom.....</a>	<a href="#">131</a>
137	<a href="#">13.7 TPM_StirRandom.....</a>	<a href="#">132</a>
138	<a href="#">13.8 TPM_CertifyKey.....</a>	<a href="#">133</a>
139	<a href="#">13.9 TPM_CertifyKey2.....</a>	<a href="#">138</a>
140	<a href="#">14. Endorsement Key Handling.....</a>	<a href="#">143</a>
141	<a href="#">14.1 TPM_CreateEndorsementKeyPair.....</a>	<a href="#">144</a>
142	<a href="#">14.2 TPM_CreateRevocableEK.....</a>	<a href="#">146</a>
143	<a href="#">14.3 TPM_RevokeTrust.....</a>	<a href="#">148</a>
144	<a href="#">14.4 TPM_ReadPubek.....</a>	<a href="#">149</a>
145	<a href="#">14.5 TPM_OwnerReadInternalPub.....</a>	<a href="#">150</a>
146	<a href="#">15. Identity Creation and Activation.....</a>	<a href="#">152</a>
147	<a href="#">15.1 TPM_MakeIdentity.....</a>	<a href="#">152</a>
148	<a href="#">15.2 TPM_ActivateIdentity.....</a>	<a href="#">156</a>
149	<a href="#">16. Integrity Collection and Reporting.....</a>	<a href="#">159</a>
150	<a href="#">16.1 TPM_Extend.....</a>	<a href="#">160</a>
151	<a href="#">16.2 TPM_PCRRead.....</a>	<a href="#">162</a>
152	<a href="#">16.3 TPM_Quote.....</a>	<a href="#">163</a>
153	<a href="#">16.4 TPM_PCR_Reset.....</a>	<a href="#">165</a>
154	<a href="#">16.5 TPM_Quote2.....</a>	<a href="#">167</a>
155	<a href="#">17. Changing AuthData.....</a>	<a href="#">170</a>
156	<a href="#">17.1 TPM_ChangeAuth.....</a>	<a href="#">170</a>
157	<a href="#">17.2 TPM_ChangeAuthOwner.....</a>	<a href="#">174</a>
158	<a href="#">18. Authorization Sessions.....</a>	<a href="#">176</a>
159	<a href="#">18.1 TPM_OIAP.....</a>	<a href="#">176</a>
160	<a href="#">18.1.1 Actions to validate an OIAP session.....</a>	<a href="#">177</a>
161	<a href="#">18.2 TPM_OSA.....</a>	<a href="#">179</a>
162	<a href="#">18.2.1 Actions to validate an OSA session.....</a>	<a href="#">182</a>
163	<a href="#">18.3 TPM_DSAP.....</a>	<a href="#">184</a>
164	<a href="#">18.4 TPM_SetOwnerPointer.....</a>	<a href="#">188</a>
165	<a href="#">19. Delegation Commands.....</a>	<a href="#">190</a>
166	<a href="#">19.1 TPM_Delegate_Manage.....</a>	<a href="#">190</a>
167	<a href="#">19.2 TPM_Delegate_CreateKeyDelegation.....</a>	<a href="#">194</a>
168	<a href="#">19.3 TPM_Delegate_CreateOwnerDelegation.....</a>	<a href="#">197</a>
169	<a href="#">19.4 TPM_Delegate_LoadOwnerDelegation.....</a>	<a href="#">200</a>
170	<a href="#">19.5 TPM_Delegate_ReadTable.....</a>	<a href="#">204</a>

171	<a href="#">19.6 TPM_Delegate_UpdateVerification.....</a>	<a href="#">206</a>
172	<a href="#">19.7 TPM_Delegate_VerifyDelegation.....</a>	<a href="#">209</a>
173	<a href="#">20. Non-volatile Storage.....</a>	<a href="#">211</a>
174	<a href="#">20.1 TPM_NV_DefineSpace.....</a>	<a href="#">212</a>
175	<a href="#">20.2 TPM_NV_WriteValue.....</a>	<a href="#">216</a>
176	<a href="#">20.3 TPM_NV_WriteValueAuth.....</a>	<a href="#">220</a>
177	<a href="#">20.4 TPM_NV_ReadValue.....</a>	<a href="#">222</a>
178	<a href="#">20.5 TPM_NV_ReadValueAuth.....</a>	<a href="#">225</a>
179	<a href="#">21. Session Management.....</a>	<a href="#">227</a>
180	<a href="#">21.1 TPM_KeyControlOwner.....</a>	<a href="#">227</a>
181	<a href="#">21.2 TPM_SaveContext.....</a>	<a href="#">230</a>
182	<a href="#">21.3 TPM_LoadContext.....</a>	<a href="#">233</a>
183	<a href="#">22. Eviction.....</a>	<a href="#">235</a>
184	<a href="#">22.1 TPM_FlushSpecific.....</a>	<a href="#">236</a>
185	<a href="#">23. Timing Ticks.....</a>	<a href="#">238</a>
186	<a href="#">23.1 TPM_GetTicks.....</a>	<a href="#">238</a>
187	<a href="#">23.2 TPM_TickStampBlob.....</a>	<a href="#">239</a>
188	<a href="#">24. Transport Sessions.....</a>	<a href="#">242</a>
189	<a href="#">24.1 TPM_EstablishTransport.....</a>	<a href="#">242</a>
190	<a href="#">24.2 TPM_ExecuteTransport.....</a>	<a href="#">246</a>
191	<a href="#">24.3 TPM_ReleaseTransportSigned.....</a>	<a href="#">253</a>
192	<a href="#">25. Monotonic Counter.....</a>	<a href="#">256</a>
193	<a href="#">25.1 TPM_CreateCounter.....</a>	<a href="#">256</a>
194	<a href="#">25.2 TPM_IncrementCounter.....</a>	<a href="#">258</a>
195	<a href="#">25.3 TPM_ReadCounter.....</a>	<a href="#">260</a>
196	<a href="#">25.4 TPM_ReleaseCounter.....</a>	<a href="#">261</a>
197	<a href="#">25.5 TPM_ReleaseCounterOwner.....</a>	<a href="#">263</a>
198	<a href="#">26. DAA commands.....</a>	<a href="#">265</a>
199	<a href="#">26.1 TPM_DAA_Join.....</a>	<a href="#">265</a>
200	<a href="#">26.2 TPM_DAA_Sign.....</a>	<a href="#">283</a>
201	<a href="#">27. Deprecated commands.....</a>	<a href="#">294</a>
202	<a href="#">27.1 Key commands.....</a>	<a href="#">295</a>
203	<a href="#">27.1.1 TPM_EvictKey.....</a>	<a href="#">295</a>
204	<a href="#">27.1.2 TPM_Terminate_Handle.....</a>	<a href="#">296</a>
205	<a href="#">27.2 Context management.....</a>	<a href="#">298</a>
206	<a href="#">27.2.1 TPM_SaveKeyContext.....</a>	<a href="#">298</a>
207	<a href="#">27.2.2 TPM_LoadKeyContext.....</a>	<a href="#">300</a>

208	<a href="#">27.2.3 TPM_SaveAuthContext.....</a>	<a href="#">301</a>
209	<a href="#">27.2.4 TPM_LoadAuthContext.....</a>	<a href="#">302</a>
210	<a href="#">27.3 DIR commands.....</a>	<a href="#">304</a>
211	<a href="#">27.3.1 TPM_DirWriteAuth.....</a>	<a href="#">305</a>
212	<a href="#">27.3.2 TPM_DirRead.....</a>	<a href="#">307</a>
213	<a href="#">27.4 Change Auth.....</a>	<a href="#">308</a>
214	<a href="#">27.4.1 TPM_ChangeAuthAsymStart.....</a>	<a href="#">309</a>
215	<a href="#">27.4.2 TPM_ChangeAuthAsymFinish.....</a>	<a href="#">312</a>
216	<a href="#">27.5 TPM_Reset.....</a>	<a href="#">315</a>
217	<a href="#">27.6 TPM_OwnerReadPubek.....</a>	<a href="#">316</a>
218	<a href="#">27.7 TPM_DisablePubekRead.....</a>	<a href="#">317</a>
219	<a href="#">27.8 TPM_LoadKey.....</a>	<a href="#">318</a>
220	<a href="#">28. Deleted Commands.....</a>	<a href="#">322</a>
221	<a href="#">28.1 TPM_GetCapabilitySigned.....</a>	<a href="#">323</a>
222	<a href="#">28.2 TPM_GetOrdinalAuditStatus.....</a>	<a href="#">324</a>
223	<a href="#">28.3 TPM_CertifySelfTest.....</a>	<a href="#">325</a>
224	<a href="#">28.4 TPM_GetAuditEvent.....</a>	<a href="#">327</a>
225	<a href="#">28.5 TPM_GetAuditEventSigned.....</a>	<a href="#">328</a>
226		
227	End of Introduction do not delete	

## 228**1. Scope and Audience**

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

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

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

## 236**1.1 Key words**

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

## 240**1.2 Action Order**

241 1. The order of ordinal actions is advisory.

242In particular, the order in which errors are checked is vendor specific. The TPM SHOULD  
243check for error conditions as much as possible before executing actions that alter the  
244TPM state.

245See also Part 2 “Return Codes” for a discussion of error code requirements.

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

249For example, authorized commands terminate authorization sessions on error, since the  
250response cannot roll the nonce. However, if the incoming session handle parameter cannot  
251be parsed, the session cannot be terminated.

252

253

### 254 **1.3 Statement Type**

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

263 For example:

#### 264 **Start of informative comment:**

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

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

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

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

#### 271 **End of informative comment.**

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

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

## 276**2. Description and TODO**

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



### 281 3. Admin Startup and State

#### 282 **Start of informative comment:**

283 This section is the commands that start a TPM.

#### 284 **End of informative comment.**

### 285 3.1 TPM\_Init

#### 286 **Start of informative comment:**

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

291 TPM\_Init puts the TPM into a state where it waits for the command TPM\_Startup (which  
292 specifies the type of initialization that is required).

#### 293 **End of informative comment.**

#### 294 **Definition**

295 `TPM_Init();`

296

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

#### 299 **Parameters**

300 None

#### 301 **Description**

302 1. The TPM\_Init signal indicates to the TPM that platform initialization is taking place. The  
303 TPM SHALL set the TPM into a state such that the only legal command to receive after  
304 the TPM\_Init is the TPM\_Startup command. The TPM\_Startup will further indicate to the  
305 TPM how to handle and initialize the TPM resources.

306 2. The platform design MUST be that the TPM is not the only component undergoing  
307 initialization. If the TPM\_Init signal forces the TPM to perform initialization then the  
308 platform MUST ensure that ALL components of the platform receive an initialization  
309 signal. This is to prevent an attacker from causing the TPM to initialize to a state where  
310 various masquerades are allowable. For instance, on a PC causing the TPM to initialize  
311 and expect measurements in PCR0 but the remainder of the platform does not initialize.

312 3. The design of the TPM MUST be such that the ONLY mechanism that signals TPM\_Init  
313 also signals initialization to the other platform components.

#### 314 **Actions**

315 1. The TPM sets TPM\_STANY\_FLAGS -> postInitialise to TRUE.

## 316**3.2 TPM\_Startup**

### 317**Start of informative comment:**

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

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

325The TPM can startup in three different modes:

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

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

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

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

### 335**End of informative comment.**

### 336**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

### 337**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

### 338**Description**

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

340 1. If the TPM is in failure mode

- 341 a. TPM\_STANY\_FLAGS -> postInitialize is still set to FALSE  
342 b. The TPM returns TPM\_FAILEDSELFTEST

### 343 Actions

- 344 1. If TPM\_STANY\_FLAGS -> postInitialise is FALSE,  
345 a. Then the TPM MUST return TPM\_INVALID\_POSTINIT, and exit this capability  
346 2. If stType = TPM\_ST\_CLEAR  
347 a. Ensure that sessions associated with resources TPM\_RT\_CONTEXT, TPM\_RT\_AUTH,  
348 TPM\_RT\_DAA\_TPM, and TPM\_RT\_TRANS are invalidated  
349 b. Reset TPM\_STCLEAR\_DATA -> PCR[] values to each correct default value  
350 i. pcrReset is FALSE, set to 0x00..00  
351 ii. pcrReset is TRUE, set to 0xFF..FF  
352 c. Set the following TPM\_STCLEAR\_FLAGS to their default state  
353 i. PhysicalPresence  
354 ii. PhysicalPresenceLock  
355 iii. disableForceClear  
356 d. The TPM MAY initialize auditDigest to all zeros  
357 i. If not initialized to all zeros, the TPM SHALL ensure that auditDigest contains  
358 a valid value.  
359 ii. If initialization fails, the TPM SHALL set auditDigest to all zeros and SHALL set  
360 the internal TPM state so that the TPM returns TPM\_FAILEDSELFTEST to all  
361 subsequent commands.  
362 e. The TPM SHALL set TPM\_STCLEAR\_FLAGS -> deactivated to the same state as  
363 TPM\_PERMANENT\_FLAGS -> deactivated  
364 f. The TPM MUST set the TPM\_STANY\_DATA fields to:  
365 i. TPM\_STANY\_DATA->contextNonceSession is set to all zeros  
366 ii. TPM\_STANY\_DATA->contextCount is set to 0  
367 iii. TPM\_STANY\_DATA->contextList is set to 0  
368 g. The TPM MUST set TPM\_STCLEAR\_DATA fields to:  
369 i. Invalidate contextNonceKey  
370 ii. countID to zero  
371 iii. ownerReference to TPM\_KH\_OWNER  
372 h. The TPM MUST set the following TPM\_STCLEAR\_FLAGS to  
373 i. bGlobalLock to FALSE  
374 i. Determine which keys should remain in the TPM  
375 i. For each key that has a valid preserved value in the TPM

- 376           (1)    if parentPCRStatus is TRUE then call TPM\_FlushSpecific(keyHandle)  
377           (2)    if isVolatile is TRUE then call TPM\_FlushSpecific(keyHandle)  
378           ii. Keys under control of the OwnerEvict flag MUST stay resident in the TPM
3793. If stType = TPM\_ST\_STATE
- 380   a. If the TPM has no state to restore, the TPM MUST set the internal state such that it  
381   returns TPM\_FAILEDSELFTEST to all subsequent commands.
- 382   b. The TPM MAY determine for each session type (authorization, transport, DAA, ...) to  
383   release or maintain the session information. The TPM reports how it manages sessions  
384   in the TPM\_GetCapability command.
- 385   c. The TPM SHALL take all necessary actions to ensure that all PCRs contain valid  
386   preserved values. If the TPM is unable to successfully complete these actions, it SHALL  
387   enter the TPM failure mode.
- 388       i. For resettable PCR the TPM MUST set the value of TPM\_STCLEAR\_DATA ->  
389       PCR[] to the resettable PCR default value. The TPM MUST NOT restore a resettable  
390       PCR to a preserved value
- 391   d. The TPM MAY initialize auditDigest to all zeros.
- 392       i. Otherwise, the TPM SHALL take all actions necessary to ensure that  
393       auditDigest contains a valid value. If the TPM is unable to successfully complete  
394       these actions, the TPM SHALL initialize auditDigest to all zeros and SHALL set the  
395       internal state such that the TPM returns TPM\_FAILEDSELFTEST to all  
396       subsequent commands.
- 397   e. The TPM MUST restore the following flags to their preserved states:
- 398       i. All values in TPM\_STCLEAR\_FLAGS
- 399       ii. All values in TPM\_STCLEAR\_DATA
- 400   f. The TPM MUST restore all keys that have a valid preserved value.
- 401   g. The TPM resumes normal operation. If the TPM is unable to resume normal  
402   operation, it SHALL enter the TPM failure mode.
4034. If stType = TPM\_ST\_DEACTIVATED
- 404   a. Invalidate sessions
- 405       i. Ensure that all resources associated with saved and active sessions are  
406       invalidated
- 407       b. Set the TPM\_STCLEAR\_FLAGS to their default state.
- 408   c. Set TPM\_STCLEAR\_FLAGS -> deactivated to TRUE
4095. The TPM MUST ensure that state associated with TPM\_SaveState is invalidated
4106. The TPM MUST set TPM\_STANY\_FLAGS -> postInitialise to FALSE

**4113.3 TPM\_SaveState****412Start of informative comment:**

413This warns a TPM to save some state information.

414If the relevant shielded storage is non-volatile, this command need have no effect.

415If the relevant shielded storage is volatile and the TPM alone is unable to detect the loss of  
416external power in time to move data to non-volatile memory, this command should be  
417presented before the TPM enters a low or no power state.

418Resettable PCRs are tied to platform state that does not survive a sleep state. If the PCRs  
419did not reset, they would falsely indicate that the platform state was already present when it  
420came out of sleep. Since some setup is required first, there would be a gap where PCRs  
421indicated the wrong state. Therefore, the PCRs must be recreated.

422Any loaded keys may be preserved. Keys with parentPCRStatus TRUE are not given priority  
423because of security concerns. Rather, since the key might be part of a storage tree that  
424requires PCR value transitions, it might not be directly loadable after  
425TPM\_Startup(ST\_STATE). For a TPM implementation that does not save all loaded keys, the  
426platform should issue a TPM\_SaveContext / TPM\_LoadContext sequence for those loaded  
427keys. contextNonceKey will be restored, guaranteeing that the saved key context can be  
428restored.

**429End of informative comment.****430Incoming 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.

**431Outgoing 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.

**432Description**

4331. Preserved values MUST be non-volatile.

4342. If data is never stored in a volatile medium, that data MAY be used as preserved data. In  
435 such cases, no explicit action may be required to preserve that data.

4363. If an explicit action is required to preserve data, it MUST be possible for the TPM to  
437 determine whether preserved data is valid.
4384. If a parameter mirrored by any preserved value is altered, all preserved values MUST be  
439 declared invalid.
4405. The TPM MAY declare all preserved values invalid in response to any command other  
441 than TPM\_Init.

## 442Actions

4431. Store TPM\_STCLEAR\_DATA -> PCR contents except for
- 444 a. If the PCR attribute pcrReset is TRUE
- 445 b. Any platform identified debug PCR
4462. The auditDigest MUST be handled according to the audit requirements as reported by  
447 TPM\_GetCapability.
- 448 a. If the ordinalAuditStatus is TRUE for the TPM\_SaveState ordinal and the auditDigest  
449 is being stored in the saved state, the saved auditDigest MUST include the  
450 TPM\_SaveState input parameters and MUST NOT include the output parameters.
4513. All values in TPM\_STCLEAR\_DATA MUST be preserved.
4524. All values in TPM\_STCLEAR\_FLAGS MUST be preserved.
4535. The contents of any key that is currently loaded SHOULD be preserved if the key's  
454 parentPCRStatus indicator is TRUE.
4556. The contents of any key that has TPM\_KEY\_CONTROL\_OWNER\_EVICT set MUST be  
456 preserved
4577. The contents of any key that is currently loaded MAY be preserved.
4588. The contents of sessions (authorization, transport, DAA, etc.) MAY be preserved as  
459 reported by TPM\_GetCapability.

## 4604. Admin Testing

### 4614.1 TPM\_SelfTestFull

#### 462Start of informative comment:

463TPM\_SelfTestFull tests all of the TPM capabilities.

464Unlike TPM\_ContinueSelfTest, which may optionally return immediately and then perform  
465the tests, TPM\_SelfTestFull always performs the tests and then returns success or failure.

#### 466End of informative comment.

#### 467Incoming 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

#### 468Outgoing 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

#### 469Actions

4701. TPM\_SelfTestFull SHALL cause a TPM to perform self-test of each TPM internal function.

471 a. If the self-test succeeds, return TPM\_SUCCESS.

472 b. If the self-test fails, return TPM\_FAILEDSELFTEST.

4732. Failure of any test results in overall failure, and the TPM goes into failure mode.

4743. If the TPM has not executed the action of TPM\_ContinueSelfTest, the TPM

475 a. MAY perform the full self-test.

476 b. MAY return TPM\_NEEDS\_SELFTEST.

## 4774.2 TPM\_ContinueSelfTest

### 478Start of informative comment:

479TPM\_ContinueSelfTest informs the TPM that it should complete the self-test of all TPM  
480functions.

481The TPM may return success immediately and then perform the self-test, or it may perform  
482the self-test and then return success or failure.

### 483End of informative comment.

### 484Incoming 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

### 485Outgoing 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

### 486Description

4871. Prior to executing the actions of TPM\_ContinueSelfTest, if the TPM receives a command  
488 C1 that uses an untested TPM function, the TPM MUST take one of these actions:

489 a. The TPM MAY return TPM\_NEEDS\_SELFTEST

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

492 ii. The TPM does not execute C1.

493 iii. The caller MUST issue TPM\_ContinueSelfTest before re-issuing the command  
494 C1.

495 (1) If the TPM permits TPM\_SelfTestFull prior to completing the actions of  
496 TPM\_ContinueSelfTest, the caller MAY issue TPM\_SelfTestFull rather than  
497 TPM\_ContinueSelfTest.

498 b. The TPM MAY return TPM\_DOING\_SELFTEST

499 i. This indicates that the TPM is doing the actions of TPM\_ContinueSelfTest  
500 implicitly, as if the TPM\_ContinueSelfTest command had been issued.

501 ii. The TPM does not execute C1.



- 109  
110
- 502           iii. The caller MUST wait for the actions of TPM\_ContinueSelfTest to complete  
503           before reissuing the command C1.
- 504    c. The TPM MAY return TPM\_SUCCESS or an error code associated with C1.
- 505           i. This indicates that the TPM has completed the actions of  
506           TPM\_ContinueSelfTest and has completed the command C1.
- 507           ii. The error code MAY be TPM\_FAILEDSELFTEST.

## 508 Actions

- 509 1. If TPM\_PERMANENT\_FLAGS -> FIPS is TRUE or TPM\_PERMANENT\_FLAGS -> TPMpost  
510    is TRUE
- 511    a. The TPM MUST run all self-tests
- 512 2. Else
- 513    a. The TPM MUST complete all self-tests that are outstanding
- 514           i. Instead of completing all outstanding self-tests the TPM MAY run all self-tests
- 515 3. The TPM either
- 516    a. MAY immediately return TPM\_SUCCESS
- 517           i. When TPM\_ContinueSelfTest finishes execution, it MUST NOT respond to the  
518           caller with a return code.
- 519    b. MAY complete the self-test and then return TPM\_SUCCESS or  
520       TPM\_FAILEDSELFTEST.

### 5214.3 TPM\_GetTestResult

#### 522Start of informative comment:

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

#### 527End of informative comment.

#### 528Incoming Operands and Sizes

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

#### 529Outgoing Operands and Sizes

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

#### 530Description

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

#### 533Actions

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

5362. The information MUST NOT contain any data that uniquely identifies an individual TPM.

## 5375. Admin Opt-in

### 5385.1 TPM\_SetOwnerInstall

#### 539Start of informative comment:

540When enabled but without an owner this command sets the PERMANENT flag that allows or  
541disallows the ability to insert an owner.

#### 542End of informative comment.

#### 543Incoming 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.

#### 544Outgoing 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

#### 545Action

5461. If the TPM has a current owner, this command immediately returns with  
547 TPM\_SUCCESS.

5482. The TPM validates the assertion of physical presence. The TPM then sets the value of  
549 TPM\_PERMANENT\_FLAGS -> ownership to the value in state.

## 5505.2 TPM\_OwnerSetDisable

### 551Start of informative comment:

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

### 553End of informative comment.

### 554Incoming Operands and Sizes

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

### 555Outgoing Operands and Sizes

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

### 556Action

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

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

## 5615.3 TPM\_PhysicalEnable

### 562Start of informative comment:

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

### 564End of informative comment.

### 565Incoming Operands and Sizes

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

### 566Outgoing Operands and Sizes

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

### 567Action

5681. Validate that physical presence is being asserted, if not return TPM\_BAD\_PRESENCE

5692. The TPM SHALL set the TPM\_PERMANENT\_FLAGS.disable value to FALSE.

## 5705.4 TPM\_PhysicalDisable

### 571Start of informative comment:

572Sets the PERMANENT disable flag to TRUE using physical presence as authorization

### 573End of informative comment.

### 574Incoming 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

### 575Outgoing 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

### 576Action

5771. Validate that physical presence is being asserted, if not return TPM\_BAD\_PRESENCE

5782. The TPM SHALL set the TPM\_PERMANENT\_FLAGS.disable value to TRUE.

## 579**5.5 TPM\_PhysicalSetDeactivated**

### 580**Start of informative comment:**

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

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

### 583**End of informative comment.**

### 584**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.

### 585**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

### 586**Action**

5871. Validate that physical presence is being asserted, if not return TPM\_BAD\_PRESENCE

5882. The TPM SHALL set the TPM\_PERMANENT\_FLAGS.deactivated flag to the value in the  
589 state parameter.

## 5905.6 TPM\_SetTempDeactivated

### 591Start of informative comment:

592This command allows the operator of the platform to deactivate the TPM until the next boot  
593of the platform.

594This command requires operator authentication. The operator can provide the  
595authentication by either the assertion of physical presence or presenting the operator  
596AuthData value.

### 597End of informative comment.

### 598Incoming 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

### 599Outgoing 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.

### 600Action

6011. If tag = TPM\_TAG\_RQU\_AUTH1\_COMMAND

602 a. If TPM\_PERMANENT\_FLAGS -> operator is FALSE return TPM\_NOOPERATOR

603 b. Validate command and parameters using operatorAuth, on error return  
604 TPM\_AUTHFAIL



6052. Else

606 a. If physical presence is not asserted the TPM MUST return TPM\_BAD\_PRESENCE

6073. The TPM SHALL set the TPM\_STCLEAR\_FLAGS.deactivated flag to the value TRUE.

## 6085.7 TPM\_SetOperatorAuth

### 609Start of informative comment:

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

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

### 615End of informative comment.

### 616Incoming 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

### 617Outgoing 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

### 618Action

6191. If physical presence is not asserted the TPM MUST return TPM\_BAD\_PRESENCE

6202. The TPM SHALL set the TPM\_PERMANENT\_DATA -> operatorAuth

6213. The TPM SHALL set TPM\_PERMANENT\_FLAGS -> operator to TRUE

## 622 6. Admin Ownership

### 623 6.1 TPM\_TakeOwnership

624 **Start of informative comment:**

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

626 **End of informative comment.**

### 627 Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_TakeOwnership
4	2	2S	2	TPM_PROTOCOL_ID	protocolID	The ownership protocol in use.
5	4	3S	4	UINT32	encOwnerAuthSize	The size of the encOwnerAuth field
6	<>	4S	<>	BYTE[]	encOwnerAuth	The owner AuthData encrypted with PUBEK
7	4	5S	4	UINT32	encSrAuthSize	The size of the encSrAuth field
8	<>	6S	<>	BYTE[]	encSrAuth	The SRK AuthData encrypted with PUBEK
9	<>	7S	<>	TPM_KEY	srkParams	Structure containing all parameters of new SRK. pubKey.keyLength & encSize are both 0. This structure MAY be TPM_KEY12.
10	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for this command
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
11	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
12	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
13	20			TPM_AUTHDATA	ownerAuth	Authorization session digest for input params. HMAC key: the new ownerAuth value. See actions for validation operations

628

## 629Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_TakeOwnership
4	<>	3S	<>	TPM_KEY	srkPub	Structure containing all parameters of new SRK. srkPub.encData is set to 0. This structure MAY be TPM_KEY12.
5	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
6	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
7	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: the new ownerAuth value

## 630Description

631The type of the output srkPub MUST be the same as the type of the input srkParams, either  
632both TPM\_KEY or both TPM\_KEY12.

## 633Actions

- 6341. If TPM\_PERMANENT\_DATA -> ownerAuth is valid return TPM\_OWNER\_SET
- 6352. If TPM\_PERMANENT\_FLAGS -> ownership is FALSE return TPM\_INSTALL\_DISABLED
- 6363. If TPM\_PERMANENT\_DATA -> endorsementKey is invalid return  
637 TPM\_NO\_ENDORSEMENT
- 6384. Verify that authHandle is of type OIAP on error return TPM\_AUTHFAIL
- 6395. If protocolID is not TPM\_PID\_OWNER, the TPM MAY return TPM\_BAD\_PARAMETER
- 6406. Create A1 a TPM\_SECRET by decrypting encOwnerAuth using PRIVEK as the key  
641 a. This requires that A1 was encrypted using the PUBEK  
642 b. Validate that A1 is a length of 20 bytes, on error return TPM\_BAD\_KEY\_PROPERTY
- 6437. Validate the command and parameters using A1 and ownerAuth, on error return  
644 TPM\_AUTHFAIL
- 6458. Validate srkParams  
646 a. If srkParams -> keyUsage is not TPM\_KEY\_STORAGE return  
647 TPM\_INVALID\_KEYUSAGE  
648 b. If srkParams -> migratable is TRUE return TPM\_INVALID\_KEYUSAGE  
649 c. If srkParams -> algorithmParms -> algorithmID is NOT TPM\_ALG\_RSA return  
650 TPM\_BAD\_KEY\_PROPERTY  
651 d. If srkParams -> algorithmParms -> encScheme is NOT  
652 TPM\_ES\_RSAESOAEP\_SHA1\_MGF1 return TPM\_BAD\_KEY\_PROPERTY

163  
164  
653 e. If srkParams -> algorithmParms -> sigScheme is NOT TPM\_SS\_NONE return  
654 TPM\_BAD\_KEY\_PROPERTY  
655 f. srkParams -> algorithmParms -> parms -> keyLength MUST be greater than or equal  
656 to 2048, on error return TPM\_BAD\_KEY\_PROPERTY  
657 g. If srkParams -> algorithmParms -> parms -> exponentSize is not 0, return  
658 TPM\_BAD\_KEY\_PROPERTY  
659 h. If TPM\_PERMANENT\_FLAGS -> FIPS is TRUE  
660 i. If srkParams -> authDataUsage specifies TPM\_AUTH\_NEVER return  
661 TPM\_NOTFIPS  
6629. Generate K1 according to the srkParams, on error return TPM\_BAD\_KEY\_PROPERTY  
663 a. This includes copying PCRInfo from srkParams to K1  
66410. Create A2 a TPM\_SECRET by decrypting encSrkJAuth using the PRIVEK  
665 a. This requires A2 to be encrypted using the PUBEK  
666 b. Validate that A2 is a length of 20 bytes, on error return TPM\_BAD\_KEY\_PROPERTY  
667 c. Store A2 in K1 -> usageAuth  
66811. Store K1 in TPM\_PERMANENT\_DATA -> srk  
66912. Store A1 in TPM\_PERMANENT\_DATA -> ownerAuth  
67013. Create TPM\_PERMANENT\_DATA -> contextKey according to the rules for the algorithm  
671 in use by the TPM to save context blobs  
67214. Create TPM\_PERMANENT\_DATA -> delegateKey according to the rules for the algorithm  
673 in use by the TPM to save delegate blobs  
67415. Create TPM\_PERMANENT\_DATA -> tpmProof by using the TPM RNG  
67516. Export TPM\_PERMANENT\_DATA -> srk as srkJPub  
67617. Set TPM\_PERMANENT\_FLAGS -> readPubek to FALSE  
67718. Calculate resAuth using the newly established TPM\_PERMANENT\_DATA -> ownerAuth

## 6786.2 TPM\_OwnerClear

### 679Start of informative comment:

680The TPM\_OwnerClear command performs the clear operation under Owner authentication.  
681This command is available until the Owner executes the TPM\_DisableOwnerClear, at which  
682time any further invocation of this command returns TPM\_CLEAR\_DISABLED.

### 683End of informative comment.

### 684Incoming Operands and Sizes

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

### 685Outgoing Operands and Sizes

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

### 686Actions

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

6892. If TPM\_PERMANENT\_FLAGS -> disableOwnerClear is TRUE then return  
690 TPM\_CLEAR\_DISABLED.

6913. Unload all loaded keys.

692 a. This includes owner evict keys

- 172  
173
- 693 b. If TPM\_PERMANENT\_FLAGS -> FIPS is TRUE, the memory locations containing  
694 secret or private keys MUST be set to all zeros.
6954. The TPM MUST NOT modify the following TPM\_PERMANENT\_DATA items
- 696 a. endorsementKey  
697 b. revMajor  
698 c. revMinor  
699 d. manuMaintPub  
700 e. auditMonotonicCounter  
701 f. monotonicCounter  
702 g. pcrAttrib  
703 h. rngState  
704 i. EKReset  
705 j. lastFamilyID  
706 k. tpmDAASeed  
707 l. authDIR[0]  
708 m. daaProof  
709 n. daaBlobKey
7105. The TPM MUST invalidate the following TPM\_PERMANENT\_DATA items and any internal  
711 resources associated with these items
- 712 a. ownerAuth  
713 b. srk  
714 c. delegateKey  
715 d. delegateTable  
716 e. contextKey  
717 f. tpmProof  
718 g. operatorAuth
7196. The TPM MUST reset to manufacturing defaults the following TPM\_PERMANENT\_DATA  
720 items
- 721 a. noOwnerNVWrite MUST be set to 0  
722 b. ordinalAuditStatus  
723 c. restrictDelegate
7247. The TPM MUST invalidate or reset all fields of TPM\_STANY\_DATA
- 725 a. Nonces SHALL be reset  
726 b. Lists (e.g. contextList) SHALL be invalidated
7278. The TPM MUST invalidate or reset all fields of TPM\_STCLEAR\_DATA except the PCR's

- 728 a. Nonces SHALL be reset
- 729 b. Lists (e.g. contextList) SHALL be invalidated
- 730 c. deferredPhysicalPresence MUST be set to 0
- 7319. The TPM MUST set the following TPM\_PERMANENT\_FLAGS to their default values
- 732 a. disable
- 733 b. deactivated
- 734 c. readPubek
- 735 d. disableOwnerClear
- 736 e. disableFullDALogicInfo
- 737 f. allowMaintenance
- 738 g. readSRKPub
- 73910. The TPM MUST set the following TPM\_PERMANENT\_FLAGS
- 740 a. ownership to TRUE
- 741 b. operator to FALSE
- 742 c. maintenanceDone to FALSE
- 74311. The TPM releases all TPM\_PERMANENT\_DATA -> monotonicCounter settings
- 744 a. This includes invalidating all currently allocated counters. The result will be no
- 745 currently allocated counters and the new owner will need to allocate counters. The
- 746 actual count value will continue to increase.
- 74712. The TPM MUST deallocate all defined NV storage areas where
- 748 a. TPM\_NV\_PER\_OWNERWRITE is TRUE if nvIndex does not have the “D” bit set
- 749 b. TPM\_NV\_PER\_OWNERREAD is TRUE if nvIndex does not have the “D” bit set
- 750 c. The TPM MUST NOT deallocate any other currently defined NV storage areas.
- 75113. The TPM MUST invalidate all familyTable entries
- 75214. The TPM MUST terminate all sessions, active or saved.



## 753 6.3 TPM\_ForceClear

### 754 Start of informative comment:

755 The TPM\_ForceClear command performs the Clear operation under physical access. This  
 756 command is available until the execution of the TPM\_DisableForceClear, at which time any  
 757 further invocation of this command returns TPM\_CLEAR\_DISABLED.

758 TPM\_ForceClear can succeed even if no owner is installed. In that case, it does whatever  
 759 TPM\_OwnerClear actions that it can.

### 760 End of informative comment.

### 761 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

### 762 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

### 763 Actions

7641. The TPM SHALL check for the assertion of physical presence, if not present return  
 765 TPM\_BAD\_PRESENCE

7662. If TPM\_STCLEAR\_FLAGS -> disableForceClear is TRUE return TPM\_CLEAR\_DISABLED

7673. The TPM SHALL execute the actions of TPM\_OwnerClear (except for the TPM Owner  
 768 authentication check)

## 7696.4 TPM\_DisableOwnerClear

### 770Start of informative comment:

771The TPM\_DisableOwnerClear command disables the ability to execute the TPM\_OwnerClear  
772command permanently. Once invoked the only method of clearing the TPM will require  
773physical access to the TPM.

774After the execution of TPM\_ForceClear, ownerClear is re-enabled and must be explicitly  
775disabled again by the new TPM Owner.

### 776End of informative comment.

### 777Incoming 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.

### 778Outgoing 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.

### 779Actions

7801. The TPM verifies that the authHandle properly authorizes the owner.

7812. The TPM sets the TPM\_PERMANENT\_FLAGS -> disableOwnerClear flag to TRUE.

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

## 7856.5 TPM\_DisableForceClear

### 786Start of informative comment:

787The TPM\_DisableForceClear command disables the execution of the TPM\_ForceClear  
788command until the next startup cycle. Once this command is executed, the TPM\_ForceClear  
789is disabled until another startup cycle is run.

### 790End of informative comment.

### 791Incoming 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

### 792Outgoing 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

### 793Actions

7941. The TPM sets the TPM\_STCLEAR\_FLAGS.disableForceClear flag in the TPM that disables  
795 the execution of the TPM\_ForceClear command.

## 7966.6 TSC\_PhysicalPresence

### 797Start of informative comment:

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

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

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

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

### 812End of informative comment.

### 813Incoming 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.

### 814Outgoing 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.

### 815Actions

8161. For documentation ease, the bits break into two categories. The first is the lifetime  
817 settings and the second is the assertion settings.

818 a. Define A1 to be the lifetime settings: TPM\_PHYSICAL\_PRESENCE\_LIFETIME\_LOCK,  
819 TPM\_PHYSICAL\_PRESENCE\_HW\_ENABLE, TPM\_PHYSICAL\_PRESENCE\_CMD\_ENABLE,  
820 TPM\_PHYSICAL\_PRESENCE\_HW\_DISABLE, and  
821 TPM\_PHYSICAL\_PRESENCE\_CMD\_DISABLE

822 b. Define A2 to be the assertion settings: TPM\_PHYSICAL\_PRESENCE\_LOCK,  
823 TPM\_PHYSICAL\_PRESENCE\_PRESENT, and TPM\_PHYSICAL\_PRESENCE\_NOTPRESENT

#### 824Lifetime lock settings

8252. If any A1 setting is present

826 a. If TPM\_PERMANENT\_FLAGS -> physicalPresenceLifetimeLock is TRUE, return  
827 TPM\_BAD\_PARAMETER

828 b. If any A2 setting is present return TPM\_BAD\_PARAMETER

829 c. If both physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_HW\_ENABLE and  
830 physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_HW\_DISABLE are TRUE, return  
831 TPM\_BAD\_PARAMETER.

832 d. If both physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_CMD\_ENABLE and  
833 physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_CMD\_DISABLE are TRUE, return  
834 TPM\_BAD\_PARAMETER.

835 e. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_HW\_ENABLE is TRUE Set  
836 TPM\_PERMANENT\_FLAGS -> physicalPresenceHWEEnable to TRUE

837 f. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_HW\_DISABLE is TRUE Set  
838 TPM\_PERMANENT\_FLAGS -> physicalPresenceHWEEnable to FALSE

839 g. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_CMD\_ENABLE is TRUE, Set  
840 TPM\_PERMANENT\_FLAGS -> physicalPresenceCMDEnable to TRUE.

841 h. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_CMD\_DISABLE is TRUE, Set  
842 TPM\_PERMANENT\_FLAGS -> physicalPresenceCMDEnable to FALSE.

843 i. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_LIFETIME\_LOCK is TRUE

844 i. Set TPM\_PERMANENT\_FLAGS -> physicalPresenceLifetimeLock to TRUE

845 j. Return TPM\_SUCCESS

#### 846SW physical presence assertion

8473. If any A2 setting is present

848 a. If any A1 setting is present return TPM\_BAD\_PARAMETER

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

851 b. If TPM\_PERMANENT\_FLAGS -> physicalPresenceCMDEnable is FALSE, return  
852 TPM\_BAD\_PARAMETER

853 c. If both physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_LOCK and physicalPresence  
854 -> TPM\_PHYSICAL\_PRESENCE\_PRESENT are TRUE, return TPM\_BAD\_PARAMETER

855 d. If both physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_PRESENT and  
856 physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_NOTPRESENT are TRUE, return  
857 TPM\_BAD\_PARAMETER

858 e. If TPM\_STCLEAR\_FLAGS -> physicalPresenceLock is TRUE, return  
859 TPM\_BAD\_PARAMETER

208  
209

860 f. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_LOCK is TRUE  
861 i. Set TPM\_STCLEAR\_FLAGS -> physicalPresence to FALSE  
862 ii. Set TPM\_STCLEAR\_FLAGS -> physicalPresenceLock to TRUE  
863 iii. Return TPM\_SUCCESS  
864 g. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_PRESENT is TRUE  
865 i. Set TPM\_STCLEAR\_FLAGS -> physicalPresence to TRUE  
866 h. If physicalPresence -> TPM\_PHYSICAL\_PRESENCE\_NOTPRESENT is TRUE  
867 i. Set TPM\_STCLEAR\_FLAGS -> physicalPresence to FALSE  
868 i. Return TPM\_SUCCESS  
8694. Else // There were no A1 or A2 parameters set  
870 a. Return TPM\_BAD\_PARAMETER

## 8716.7 TSC\_ResetEstablishmentBit

### 872Start of informative comment:

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

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

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

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

### 887End of informative comment.

### 888Incoming Operands and Sizes

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

### 889Outgoing Operands and Sizes

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

### 890Actions

8911. Validate the assertion of locality 3 or locality 4

8922. Set TPM\_PERMANENT\_FLAGS -> tpmEstablished to FALSE

8933. Return TPM\_SUCCESS



## 894 7. The Capability Commands

### 895 **Start of informative comment:**

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

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

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

905 In version 1.2 with the deletion of TPM\_GetCapabilitySigned the way to obtain a signed  
906 listing of the capabilities is to create a transport session, perform TPM\_GetCapability  
907 commands to list the information and then close the transport session using  
908 TPM\_ReleaseTransportSigned.

### 909 **End of informative comment.**

910 1. The standard information provided in TPM\_GetCapability MUST NOT provide unique  
911 information

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

## 9157.1 TPM\_GetCapability

### 916Start of informative comment:

917This command returns current information regarding the TPM.

918The limitation on what can be returned in failure mode restricts the information a  
919manufacturer may return when capArea indicates TPM\_CAP\_MFR.

### 920End of informative comment.

### 921Incoming 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

### 922Outgoing 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

923

## 924 Actions

- 925 1. The TPM validates the capArea and subCap indicators. If the information is available,  
926 the TPM creates the response field and fills in the actual information.
- 927 2. The structure document contains the list of caparea and subCap values
- 928 3. If the TPM is in failure mode or limited operation mode, the TPM MUST return
- 929 a. TPM\_CAP\_VERSION
- 930 b. TPM\_CAP\_VERSION\_VAL
- 931 c. TPM\_CAP\_MFR
- 932 d. TPM\_CAP\_PROPERTY -> TPM\_CAP\_PROP\_MANUFACTURER
- 933 e. TPM\_CAP\_PROPERTY -> TPM\_CAP\_PROP\_DURATION
- 934 f. TPM\_CAP\_PROPERTY -> TPM\_CAP\_PROP\_TIS\_TIMEOUT
- 935 g. The TPM MAY return any other capability.

## 9367.2 TPM\_SetCapability

### 937Start of informative comment:

938This command sets values in the TPM.

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

### 941End of informative comment.

### 942Incoming Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	ordinal: TPM_ORD_SetCapability
4	4	2S	4	TPM_CAPABILITY_AREA	capArea	Partition of capabilities to be set
5	4	3S	4	UINT32	subCapSize	Size of subCap parameter
6	<>	4S	<>	BYTE[]	subCap	Further definition of information
7	4	5S	4	UINT32	setValueSize	The size of the value to set
8	<>	6S	<>	BYTE[]	setValue	The value to set
9	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
10	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
11	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
12	20			TPM_AUTHDATA	ownerAuth	Authorization. HMAC key: owner.usageAuth.

943**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.

944**Actions**

9451. If tag = TPM\_TAG\_RQU\_AUTH1\_COMMAND, validate the command and parameters  
946 using ownerAuth, return TPM\_AUTHFAIL on error
9472. The TPM validates the capArea and subCap indicators, including the ability to set value  
948 based on any set restrictions
9493. If the capArea and subCap indicators conform with one of the entries in the structure  
950 TPM\_CAPABILITY\_AREA (Values for TPM\_SetCapability)
- 951 a. The TPM sets the relevant flag/data to the value of setValue parameter.
9524. Else
- 953 a. Return the error code TPM\_BAD\_PARAMETER.

## 9547.3 TPM\_GetCapabilityOwner

### 955Start of informative comment:

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

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

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

### 964End of informative comment.

### 965Incoming 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.

966

### 967Outgoing 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.

## 969 **Description**

970 For  $31 \geq N \geq 0$

971 1. Bit-N of the TPM\_PERMANENT\_FLAGS structure is the Nth bit after the opening bracket  
972 in the definition of TPM\_PERMANENT\_FLAGS in the version of the specification  
973 indicated by the parameter “version”. The bit immediately after the opening bracket is  
974 the 0<sup>th</sup> bit.

975 2. Bit-N of the TPM\_STCLEAR\_FLAGS structure is the Nth bit after the opening bracket in  
976 the definition of TPM\_STCLEAR\_FLAGS in the version of the specification indicated by  
977 the parameter “version”. The bit immediately after the opening bracket is the 0<sup>th</sup> bit.

978 3. Bit-N of non\_volatile\_flags corresponds to the Nth bit in TPM\_PERMANENT\_FLAGS, and  
979 the lsb of non\_volatile\_flags corresponds to bit0 of TPM\_PERMANENT\_FLAGS

980 4. Bit-N of volatile\_flags corresponds to the Nth bit in TPM\_STCLEAR\_FLAGS, and the lsb  
981 of volatile\_flags corresponds to bit0 of TPM\_STCLEAR\_FLAGS

## 982 **Actions**

983 1. The TPM validates that the TPM Owner authorizes the command.

984 2. The TPM creates the parameter non\_volatile\_flags by setting each bit to the same state  
985 as the corresponding bit in TPM\_PERMANENT\_FLAGS. Bits in non\_volatile\_flags for  
986 which there is no corresponding bit in TPM\_PERMANENT\_FLAGS are set to zero.

987 3. The TPM creates the parameter volatile\_flags by setting each bit to the same state as the  
988 corresponding bit in TPM\_STCLEAR\_FLAGS. Bits in volatile\_flags for which there is no  
989 corresponding bit in TPM\_STCLEAR\_FLAGS are set to zero.

990 4. The TPM generates the parameter “version”.

991 5. The TPM returns non\_volatile\_flags, volatile\_flags and version to the caller.

## 9928. Auditing

### 9938.1 Audit Generation

#### 994Start of informative comment:

995The TPM generates an audit event in response to the TPM successfully executing a  
996command that has the audit flag set to TRUE for that command ordinal.

997The TPM maintains an extended value for all audited operations.

#### 998End of informative comment.

#### 999Description

10001. The TPM extends the audit digest whenever the ordinalAuditStatus is TRUE for the  
1001 ordinal about to be executed.

10022. The TPM extends the audit digest only when a command is successfully executed.

1003 a. If the ordinal is unknown, unimplemented, or cannot be determined, no auditing is  
1004 performed.

10053. Corner cases

1006 a. TPM\_SaveState: Only the input parameters are audited, and the audit occurs before  
1007 the state is saved. If an error occurs while or after the state is saved, the audit still  
1008 occurs.

1009 b. TPM\_SetOrdinalAuditStatus: In the case where the ordinalToAudit is  
1010 TPM\_ORD\_SetOrdinalAuditStatus, audit is based on the initial state, not the final  
1011 state.

#### 1012Actions

1013The TPM will execute the ordinal and perform auditing in the following manner:

10141. Execute command

1015 a. Execution implies the performance of the listed actions for the ordinal.

10162. If the command will return TPM\_SUCCESS

1017 a. If TPM\_STANY\_DATA -> auditDigest is all zeros

1018 i. Increment TPM\_PERMANENT\_DATA -> auditMonotonicCounter by 1

1019 b. Create A1 a TPM\_AUDIT\_EVENT\_IN structure

1020 i. Set A1 -> inputParms to the digest of the input parameters from the  
1021 command

1022 1. Digest value according to the HMAC digest rules of the "above the  
1023 line" parameters (i.e. the first HMAC digest calculation).

1024 ii. Set A1 -> auditCount to TPM\_PERMANENT\_DATA ->  
1025 auditMonotonicCounter



253

254

- 1026 c. Set TPM\_STANY\_DATA -> auditDigest to SHA-1 (TPM\_STANY\_DATA ->  
1027 auditDigest || A1)
- 1028 d. Create A2 a TPM\_AUDIT\_EVENT\_OUT structure
- 1029 i. Set A2 -> outputParms to the digest of the output parameters from the  
1030 command
- 1031 1. Digest value according to the HMAC digest rules of the "above the  
1032 line" parameters (i.e. the first HMAC digest calculation).
- 1033 ii. Set A2 -> auditCount to TPM\_PERMANENT\_DATA ->  
1034 auditMonotonicCounter
- 1035 e. Set TPM\_STANY\_DATA -> auditDigest to SHA-1 (TPM\_STANY\_DATA ->  
1036 auditDigest || A2)
- 1037

## 1038**8.2 Effect of audit failing**

### 1039**Start of informative comment:**

1040The TPM audit process could have an internal error when attempting to audit a command.  
1041To indicate the audit failure, the TPM will return TPM\_AUDITFAIL\_SUCCESSFUL. This new  
1042functionality changes the 1.1 TPM functionality when this condition occurs.

1043Since no audit occurs if the command fails, The TPM\_AUDITFAIL\_UNSUCCESSFUL return  
1044code is no longer used.

### 1045**End of informative comment.**

10461. When, in performing the audit process, the TPM has an internal failure (unable to write,  
1047 SHA-1 failure etc.) the TPM MUST set the internal TPM state such that the TPM returns  
1048 the TPM\_FAILEDSELFTEST error on subsequent attempts to execute a command.

10492. The return code for the command uses the following rules

- 1050 a. Command result success, audit success -> return TPM\_SUCCESS
- 1051 b. Command result failure, no audit -> return command result failure
- 1052 c. Command result success, audit failure -> return TPM\_AUDITFAIL\_SUCCESSFUL

10533. If the TPM is permanently nonrecoverable after an audit failure, then the TPM MUST  
1054 always return TPM\_FAILEDSELFTEST for every command other than  
1055 TPM\_GetTestResult. This state must persist regardless of power cycling, the execution  
1056 of TPM\_Init or any other actions.

### 1057 **8.3 TPM\_GetAuditDigest**

#### 1058 **Start of informative comment:**

1059 This returns the current audit digest. The external audit log has the responsibility to track  
1060 the parameters that constitute the audit digest.

1061 This value may be unique to an individual TPM. The value however will be changing at a  
1062 rate set by the TPM Owner. Those attempting to use this value may find it changing without  
1063 their knowledge. This value represents a very poor source of tracking uniqueness.

#### 1064 **End of informative comment.**

#### 1065 **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

#### 1066 **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.

#### 1067 **Description**

1068 1. This command is never audited.

#### 1069 **Actions**

1070 1. The TPM sets auditDigest to TPM\_STANY\_DATA -> auditDigest

1071 2. The TPM sets counterValue to TPM\_PERMANENT\_DATA -> auditMonotonicCounter

1072 3. The TPM creates an ordered list of audited ordinals. The list starts at startOrdinal listing  
1073 each ordinal that is audited.

1074 a. If startOrdinal is 0 then the first ordinal that could be audited would be TPM\_OIAP  
1075 (ordinal 0x0000000A)

1076     b. The next ordinal would be TPM\_OSAP (ordinal 0x0000000B)

10774. If the ordered list does not fit in the output buffer the TPM sets more to TRUE

10785. Return TPM\_STANY\_DATA -> auditDigest as auditDigest

**10798.4 TPM\_GetAuditDigestSigned****1080Start of informative comment:**

1081The signing of the audit log returns the entire digest value and the list of currently audited  
1082commands.

1083The inclusion of the list of audited commands as an atomic operation is to tie the current  
1084digest value with the list of commands that are being audited.

**1085Note to future architects**

1086When auditing functionality is active in a TPM, it may seem logical to remove this ordinal  
1087from the active set of ordinals as the signing functionality of this command could be  
1088handled in a signed transport session. While true, this command has a secondary affect  
1089also, resetting the audit log digest. As the reset requires TPM Owner authentication, there  
1090must be some way in this command to reflect the TPM Owner wishes. By requiring that a  
1091TPM Identity key be the only key that can sign and reset, the TPM Owner's authentication is  
1092implicit in the execution of the command (TPM Identity Keys are created and controlled by  
1093the TPM Owner only). Hence, while one might want to remove an ordinal this is not one that  
1094can be removed if auditing is functional.

**1095End of informative comment.****1096Incoming 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.

## 1097Outgoing 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.

## 1098Actions

10991. Validate the AuthData and parameters using keyAuth, return TPM\_AUTHFAIL on error
11002. Validate that keyHandle -> keyUsage is TPM\_KEY\_SIGNING, TPM\_KEY\_IDENTITY or TPM\_KEY\_LEGACY, if not return TPM\_INVALID\_KEYUSAGE
11023. The TPM validates that the key pointed to by keyHandle has a signature scheme of TPM\_SS\_RSASSAPKCS1v15\_SHA1 or TPM\_SS\_RSASSAPKCS1v15\_INFO, return TPM\_INVALID\_KEYUSAGE on error
11054. Create D1 a TPM\_SIGN\_INFO structure and set the structure defaults
  - 1106 a. Set D1 -> fixed to "ADIG"
  - 1107 b. Set D1 -> replay to antiReplay
  - 1108 c. Create D3 a list of all audited ordinals as defined in the TPM\_GetAuditDigest
  - 1109 UINT32[] ordList outgoing parameter
  - 1110 d. Create D4 the SHA-1 of D3
  - 1111 e. Set auditDigest to TPM\_STANY\_DATA -> auditDigest
  - 1112 f. Set counterValue to TPM\_PERMANENT\_DATA -> auditMonotonicCounter
  - 1113 g. Create D2 the concatenation of auditDigest || counterValue || D4
  - 1114 h. Set D1 -> data to D2
  - 1115 i. Create a digital signature of the SHA-1 of D1 by using the signature scheme for
  - 1116 keyHandle
  - 1117 j. Set ordinalDigest to D4
11185. If closeAudit == TRUE

280

281

- 1119     a. If keyHandle->keyUsage is TPM\_KEY\_IDENTITY
- 1120             i. TPM\_STANY\_DATA -> auditDigest MUST be set to all zeros.
- 1121     b. Else
- 1122             i. Return TPM\_INVALID\_KEYUSAGE

## 11238.5 TPM\_SetOrdinalAuditStatus

### 1124Start of informative comment:

1125Set the audit flag for a given ordinal. Requires the authentication of the TPM Owner.

### 1126End of informative comment.

### 1127Incoming 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.

### 1128Outgoing 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.

### 1129Actions

11301. Validate the AuthData to execute the command and the parameters

11312. Validate that the ordinal points to a valid TPM ordinal, return TPM\_BADINDEX on error

1132 a. Valid TPM ordinal means an ordinal that the TPM implementation supports

11333. Set the non-volatile flag associated with ordinalToAudit to the value in auditState



## 11349. Administrative Functions - Management

### 11359.1 TPM\_FieldUpgrade

#### 1136Start of informative comment:

1137The TPM needs a mechanism to allow for updating the protected capabilities once a TPM is  
1138in the field. Given the varied nature of TPM implementations there will be numerous  
1139methods of performing an upgrade of the protected capabilities. This command, when  
1140implemented, provides a manufacturer specific method of performing the upgrade.

1141The manufacturer can determine, within the listed requirements, how to implement this  
1142command. The command may be more than one command and actually a series of  
1143commands.

1144The IDL definition is to create an ordinal for the command. However, the remaining  
1145parameters are manufacturer specific.

1146The policy to determine when it is necessary to perform the actions of TPM\_RevokeTrust is  
1147outside the TPM spec and determined by other TCG workgroups.

1148TPM\_FieldUpgrade is gated by either owner authorization or deferred assertion of Physical  
1149Presence (via the TPM\_STCLEAR\_DATA -> deferredPhysicalPresence ->  
1150unownedFieldUpgrade flag). This gating is acknowledgement that the entity that sets the  
1151security policy for a platform must approve field upgrade for that platform. This gating can  
1152block a global attack on TPMs when the TPME's privilege information (private key) has been  
1153compromised. For blocking to be effective in an unowned TPM, the TPM's ownership flag  
1154must be FALSE. (This prevents software from taking ownership and executing  
1155TPM\_FieldUpgrade with owner authorization.)

1156If an owner is present, field upgrade MUST be owner authorized, as the actions indicate.  
1157This prevents an attacker from using physical presence to upgrade a TPM without detection  
1158by the owner.

1159The advantages of deferred assertion of Physical Presence are that it:

- 1160 • permits a TPM to be upgraded if taking ownership is undesirable or impractical.
- 1161 • permits a TPM to be upgraded in the OS environment (where Physical Presence
- 1162 typically cannot be asserted), when the TPM has no owner.

1163If it is acceptable to take ownership of a TPM temporarily, an alternative to deferred  
1164assertion of Physical Presence is the process: (1) take ownership; (2) perform an owner  
1165authorized field upgrade; (3) clear the owner from the TPM.

1166There is no requirement for patch confidentiality. Confidentiality may be implemented  
1167using a manufacturer specific mechanism, and may use a global secret such as a  
1168symmetric encryption key.

1169The TPM may set the volatile deactivated flag to TRUE if a reboot is required after the field  
1170upgrade. There is no requirement to do so.

1171The TPM must check owner authorization before changing the TPM state or beginning the  
1172upgrade. This prevents a non-owner from mounting a denial-of-service attack. It is  
1173understood that a TPM may not be able to stage the entire upgrade patch inside the TPM

1174before checking owner authorization. That TPM may be forced to move the patch outside  
1175the owner authorization HMAC.

1176Ideally, if the upgrade fails (e.g., due to an authentication failure) the TPM firmware should  
1177remain unchanged. It is understood that a TPM may not be able to stage the entire upgrade  
1178patch inside the TPM for authentication before beginning the upgrade. On a failure, that  
1179TPM may be forced to roll the firmware back to a ROMed version. It may go into an upgrade  
1180failure state, where it requires a successful field upgrade before continuing.

1181If a field upgrade adds or deletes features, the security implications must be analyzed. The  
1182associated state must be set to prevent attacks. See, for example, allowMaintenance in Part  
11832. In addition, if a field upgrade adds maintenance commands, it must atomically install  
1184the manufacturer's maintenance public key.

1185**End of informative comment.**

## 1186IDL Definition

```
1187TPM_RESULT TPM_FieldUpgrade(  
1188    [in, out] TPM_AUTH* ownerAuth,  
1189    ...);
```

## 1190Type

1191This is an optional command and a TPM is not required to implement this command in any  
1192form.

## 1193Parameters

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

## 1194Description

1195The patch integrity and authenticity verification mechanisms in the TPM MUST not require  
1196the TPM to hold a global secret. The definition of global secret is a secret value shared by  
1197more than one TPM.

1198The TPME is not allowed to pre-store or use unique identifiers in the TPM for the purpose of  
1199field upgrade. The TPM MUST NOT use the endorsement key for identification or encryption  
1200in the upgrade process. The upgrade process MAY use a TPM Identity to deliver upgrade  
1201information to specific TPM's.

1202The upgrade process SHOULD only change protected capabilities. The upgrade process  
1203SHOULD NOT change shielded locations.

1204The upgrade process MUST NOT change the disabled or deactivated state from TRUE to  
1205FALSE.

1206The upgrade process SHOULD only access data in shielded locations where this data is  
1207necessary to validate the TPM Owner, validate the TPME and manipulate the blob.

1208 The execution of a TPM\_FieldUpgrade command in a TPM MUST leave the TPM in a state  
1209 that conforms to a version of TCG's TPM specification and conforms to any extant TCG-  
1210 defined credentials (certificates) that attest to that upgraded TPM.

1211 The security target used to validate this TPM MUST include this command in the TOE.

1212 If the owner authorization fails, the state of the TPM (volatile, nonvolatile, and firmware)  
1213 MUST remain unchanged. The only exception shall be the dictionary attack mitigation  
1214 state, which should process the authentication failure.

## 1215 Actions

1216 The TPM SHALL perform the following when executing the command:

1217 1. If TPM Owner is installed

1218     a. Validate the command and parameters using TPM owner authentication, return  
1219     TPM\_AUTHFAIL on error

1220 2. Else

1221     a. If TPM\_STCLEAR\_DATA -> deferredPhysicalPresence -> unownedFieldUpgrade is  
1222     FALSE return TPM\_BAD\_PRESENCE.

1223 3. Validate that the upgrade information was sent by the TPME. The validation mechanism  
1224     MUST use a strength of function that is at least the same strength of function as a  
1225     digital signature performed using a 2048 bit RSA key.

1226 4. Validate that the upgrade target is the appropriate TPM model and version.

1227 5. Process the upgrade information and update the protected capabilities

1228 6. Set the TPM\_PERMANENT\_DATA -> revMajor and TPM\_PERMANENT\_DATA -> revMinor  
1229     to the values indicated in the upgrade. The selection of the value is a manufacturer  
1230     option.

1231     a. The TPM MAY validate that the upgrade major and minor revision are monotonically  
1232     increasing.

1233     b. The TPM MAY allow upgrade with a major and minor revision that is less than  
1234     currently installed in the TPM.

1235 7. The TPM MAY set the TPM\_STCLEAR\_FLAGS.deactivated to TRUE

## 12369.2 TPM\_SetRedirection

### 1237Start of informative comment:

1238The redirection command attaches a key to a redirection receiver.

1239When making the connection to a GPIO channel the authorization restrictions are set at  
1240connection time and not for each invocation that uses the channel.

### 1241End of informative comment.

### 1242Incoming 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

### 1243Outgoing 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

1244

### 1245Action

12461. If tag == TPM\_TAG\_RQU\_AUTH1\_COMMAND

307  
308  
1247 a. Validate the command and parameters using TPM Owner authentication, on error  
1248 return TPM\_AUTHFAIL  
1249 2. if redirCmd == TPM\_REDIR\_GPIO  
1250 a. Validate that keyHandle points to a loaded key, return TPM\_INVALID\_KEYHANDLE  
1251 on error  
1252 b. Validate the key attributes specify redirection, return TPM\_BAD\_TYPE on error  
1253 c. Validate that inputDataSize is 4, return TPM\_BAD\_PARAM\_SIZE on error  
1254 d. Validate that inputData points to a valid GPIO channel, return  
1255 TPM\_BAD\_PARAMETER on error  
1256 e. Map C1 to the TPM\_GPIO\_CONFIG\_CHANNEL structure indicated by inputData  
1257 f. If C1 -> attr specifies TPM\_GPIO\_ATTR\_OWNER  
1258 i. If tag != TPM\_TAG\_RQU\_AUTH1\_COMMAND return TPM\_AUTHFAIL  
1259 g. If C1 -> attr specifies TPM\_GPIO\_ATTR\_PP  
1260 i. If TPM\_STCLEAR\_FLAGS -> physicalPresence == FALSE, then return  
1261 TPM\_BAD\_PRESENCE  
1262 h. Return TPM\_SUCCESS  
1263 3. The TPM MAY support other redirection types. These types may be specified by TCG or  
1264 provided by the manufacturer.

## 12659.3 TPM\_ResetLockValue

### 1266Start of informative comment:

1267Command that resets the TPM dictionary attack mitigation values

1268This allows the TPM owner to cancel the effect of a number of successive authorization  
1269failures. Dictionary attack mitigation is vendor specific, and the actions here are one  
1270possible implementation. The TPM may treat an authorization failure outside the mitigation  
1271time as a normal failure and not disable the command.

1272If this command itself has an authorization failure, it is blocked for the remainder of the  
1273lock out period. This prevents a dictionary attack on the owner authorization using this  
1274command.

1275It is understood that this command allows the TPM owner to perform a dictionary attack on  
1276other authorization values by alternating a trial and this command. Similarly, delegating  
1277this command allows the owner's delegate to perform a dictionary attack.

### 1278End of informative comment.

### 1279Incoming 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

### 1280Outgoing Operands and Sizes

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

**1281 Action**

- 1282 1. If TPM\_STCLEAR\_DATA -> disableResetLock is TRUE return TPM\_AUTHFAIL
- 1283     a. The internal dictionary attack mechanism will set TPM\_STCLEAR\_DATA ->
- 1284     disableResetLock to FALSE when the timeout period expires
- 1285 2. If the command and parameters validation using ownerAuth fails
- 1286     a. Set TPM\_STCLEAR\_DATA -> disableResetLock to TRUE
- 1287     b. Restart the TPM dictionary attack lock out period
- 1288     c. Return TPM\_AUTHFAIL
- 1289 3. Reset the internal TPM dictionary attack mitigation mechanism
- 1290     a. The mechanism is vendor specific and can include time outs, reboots, and other
- 1291     mitigation strategies

## 1292**10. Storage functions**

### 1293**10.1 TPM\_Seal**

#### 1294**Start of informative comment:**

1295The SEAL operation allows software to explicitly state the future “trusted” configuration that  
1296the platform must be in for the secret to be revealed. The SEAL operation also implicitly  
1297includes the relevant platform configuration (PCR-values) when the SEAL operation was  
1298performed. The SEAL operation uses the tpmProof value to BIND the blob to an individual  
1299TPM.

1300If the UNSEAL operation succeeds, proof of the platform configuration that was in effect  
1301when the SEAL operation was performed is returned to the caller, as well as the secret data.  
1302This proof may, or may not, be of interest. If the SEALED secret is used to authenticate the  
1303platform to a third party, a caller is normally unconcerned about the state of the platform  
1304when the secret was SEALED, and the proof may be of no interest. On the other hand, if the  
1305SEALED secret is used to authenticate a third party to the platform, a caller is normally  
1306concerned about the state of the platform when the secret was SEALED. Then the proof is of  
1307interest.

1308For example, if SEAL is used to store a secret key for a future configuration (probably to  
1309prove that the platform is a particular platform that is in a particular configuration), the  
1310only requirement is that that key can be used only when the platform is in that future  
1311configuration. Then there is no interest in the platform configuration when the secret key  
1312was SEALED. An example of this case is when SEAL is used to store a network  
1313authentication key.

1314On the other hand, suppose an OS contains an encrypted database of users allowed to log  
1315on to the platform. The OS uses a SEALED blob to store the encryption key for the user-  
1316database. However, the nature of SEAL is that any SW stack can SEAL a blob for any other  
1317software stack. Hence, the OS can be attacked by a second OS replacing both the SEALED-  
1318blob encryption key, and the user database itself, allowing untrusted parties access to the  
1319services of the OS. To thwart such attacks, SEALED blobs include the past SW  
1320configuration. Hence, if the OS is concerned about such attacks, it may check to see  
1321whether the past configuration is one that is known to be trusted.

1322TPM\_Seal requires the encryption of one parameter (“Secret”). For the sake of uniformity  
1323with other commands that require the encryption of more than one parameter, the string  
1324used for XOR encryption is generated by concatenating a nonce (created during the OSAP  
1325session) with the session shared secret and then hashing the result.

1326The sealed data blob does not have a protected identifier. On a platform that does not  
1327prevent unauthorized access to data, a data blob can be exchanged by a lower layer without  
1328detection. The upper layer software must take additional measures to protect the relation  
1329between its identifier of the data blob and the blob itself.

#### 1330**End of informative comment.**



**1331 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.

**1332 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.

**1333 Description**

1334 TPM\_Seal is used to encrypt private objects that can only be decrypted using TPM\_Unseal.

**1335 Actions**

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

13372. If the inDataSize is 0 the TPM returns TPM\_BAD\_PARAMETER

13383. If the keyUsage field of the key indicated by keyHandle does not have the value  
1339 TPM\_KEY\_STORAGE, the TPM must return the error code TPM\_INVALID\_KEYUSAGE.
13404. If the keyHandle points to a migratable key then the TPM MUST return the error code  
1341 TPM\_INVALID\_KEY\_USAGE.
13425. Determine the version of pcrInfo
- 1343 a. If pcrInfoSize is 0
- 1344 i. set V1 to 1
- 1345 b. Else
- 1346 i. Point X1 as TPM\_PCR\_INFO\_LONG structure to pcrInfo
- 1347 ii. If X1 -> tag is TPM\_TAG\_PCR\_INFO\_LONG
- 1348 (1) Set V1 to 2
- 1349 iii. Else
- 1350 (1) Set V1 to 1
13516. If V1 is 1 then
- 1352 a. Create S1 a TPM\_STORED\_DATA structure
13537. else
- 1354 a. Create S1 a TPM\_STORED\_DATA12 structure
- 1355 b. Set S1 -> et to 0
13568. Set S1 -> encDataSize to 0
13579. Set S1 -> encData to all zeros
135810. Set S1 -> sealInfoSize to pcrInfoSize
135911. If pcrInfoSize is not 0 then
- 1360 a. if V1 is 1 then
- 1361 i. Validate pcrInfo as a valid TPM\_PCR\_INFO structure, return TPM\_BADINDEX  
1362 on error
- 1363 ii. Set S1 -> sealInfo -> pcrSelection to pcrInfo -> pcrSelection
- 1364 iii. Create h1 the composite hash of the PCR selected by pcrInfo -> pcrSelection
- 1365 iv. Set S1 -> sealInfo -> digestAtCreation to h1
- 1366 v. Set S1 -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease
- 1367 b. else
- 1368 i. Validate pcrInfo as a valid TPM\_PCR\_INFO\_LONG structure, return  
1369 TPM\_BADINDEX on error
- 1370 ii. Set S1 -> sealInfo -> creationPCRSelection to pcrInfo -> creationPCRSelection
- 1371 iii. Set S1 -> sealInfo -> releasePCRSelection to pcrInfo -> releasePCRSelection
- 1372 iv. Set S1 -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease

334  
335

1373 v. Set S1 -> sealInfo -> localityAtRelease to pcrInfo -> localityAtRelease

1374 vi. Create h2 the composite hash of the TPM\_STCLEAR\_DATA -> PCR selected by

1375 pcrInfo -> creationPCRSelection

1376 vii. Set S1 -> sealInfo -> digestAtCreation to h2

1377 viii. Set S1 -> sealInfo -> localityAtCreation to TPM\_STANY\_FLAGS ->

1378 localityModifier

1379 12. Create a1 by decrypting encAuth according to the ADIP indicated by authHandle.

1380 13. The TPM provides NO validation of a1. Well-known values (like all zeros) are valid and

1381 possible.

1382 14. Create S2 a TPM\_SEALED\_DATA structure

1383 a. Set S2 -> payload to TPM\_PT\_SEAL

1384 b. Set S2 -> tpmProof to TPM\_PERMANENT\_DATA -> tpmProof

1385 c. Create h3 the SHA-1 of S1

1386 d. Set S2 -> storedDigest to h3

1387 e. Set S2 -> authData to a1

1388 f. Set S2 -> dataSize to inDataSize

1389 g. Set S2 -> data to inData

1390 15. Validate that the size of S2 can be encrypted by the key pointed to by keyHandle, return

1391 TPM\_BAD\_DATASIZE on error

1392 16. Create s3 the encryption of S2 using the key pointed to by keyHandle

1393 17. Set continueAuthSession to FALSE

1394 18. Set S1 -> encDataSize to the size of s3

1395 19. Set S1 -> encData to s3

1396 20. Return S1 as sealedData

## 1397**10.2 TPM\_Unseal**

### 1398**Start of informative comment:**

1399The TPM\_Unseal operation will reveal TPM\_Seal'ed data only if it was encrypted on this  
1400platform and the current configuration (as defined by the named PCR contents) is the one  
1401named as qualified to decrypt it. Internally, TPM\_Unseal accepts a data blob generated by a  
1402TPM\_Seal operation. TPM\_Unseal decrypts the structure internally, checks the integrity of  
1403the resulting data, and checks that the PCR named has the value named during TPM\_Seal.  
1404Additionally, the caller must supply appropriate AuthData for blob and for the key that was  
1405used to seal that data.

1406If the integrity, platform configuration and authorization checks succeed, the sealed data is  
1407returned to the caller; otherwise, an error is generated.

### 1408**End of informative comment.**

### 1409**Incoming Operands and Sizes**

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

## 1410 Outgoing Operands and Sizes

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

## 1411 Actions

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

14142. If the keyUsage field of the key indicated by parentHandle does not have the value  
1415 TPM\_KEY\_STORAGE, the TPM MUST return the error code TPM\_INVALID\_KEYUSAGE.

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

14194. Determine the version of inData

1420 a. If inData -> tag = TPM\_TAG\_STORED\_DATA12

1421 i. Set V1 to 2

1422 ii. Map S2 a TPM\_STORED\_DATA12 structure to inData

1423 b. Else If inData -> ver = 1.1

1424 i. Set V1 to 1

1425 ii. Map S2 a TPM\_STORED\_DATA structure to inData

1426 c. Else

1427 i. Return TPM\_BAD\_VERSION

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

14296. Validate d1

- 1430 a. d1 MUST be a TPM\_SEALED\_DATA structure
- 1431 b. d1 -> tpmProof MUST match TPM\_PERMANENT\_DATA -> tpmProof
- 1432 c. Set S2 -> encDataSize to 0
- 1433 d. Set S2 -> encData to all zeros
- 1434 e. Create h1 the SHA-1 of S2
- 1435 f. d1 -> storedDigest MUST match h1
- 1436 g. d1 -> payload MUST be TPM\_PT\_SEAL
- 1437 h. Any failure MUST return TPM\_NOTSEALED\_BLOB
- 14387. If S2 -> sealInfoSize is not 0 then
  - 1439 a. If V1 is 1 then
    - 1440 i. Validate that S2 -> pcrInfo is a valid TPM\_PCR\_INFO structure
    - 1441 ii. Create h2 the composite hash of the PCR selected by S2 -> pcrInfo ->
    - 1442 pcrSelection
  - 1443 b. If V1 is 2 then
    - 1444 i. Validate that S2 -> pcrInfo is a valid TPM\_PCR\_INFO\_LONG structure
    - 1445 ii. Create h2 the composite hash of the TPM\_STCLEAR\_DATA -> PCR selected by
    - 1446 S2 -> pcrInfo -> releasePCRSelection
    - 1447 iii. Check that S2 -> pcrInfo -> localityAtRelease for TPM\_STANY\_DATA ->
    - 1448 localityModifier is TRUE
      - 1449 (1) For example if TPM\_STANY\_DATA -> localityModifier was 2 then S2 ->
      - 1450 pcrInfo -> localityAtRelease -> TPM\_LOC\_TWO would have to be TRUE
  - 1451 c. Compare h2 with S2 -> pcrInfo -> digestAtRelease, on mismatch return
  - 1452 TPM\_WRONGPCRVAL
- 14538. The TPM MUST validate authorization to use d1 by checking that the HMAC calculation
- 1454 using d1 -> authData as the shared secret matches the dataAuth. Return
- 1455 TPM\_AUTHFAIL on mismatch.
- 14569. If V1 is 2 and S2 -> et specifies encryption (i.e. is not all zeros) then
  - 1457 a. If tag is not TPM\_TAG\_RQU\_AUTH2\_COMMAND, return TPM\_AUTHFAIL
  - 1458 b. Verify that the authHandle session type is TPM\_PID\_OSAP or TPM\_PID\_DSAP, return
  - 1459 TPM\_BAD\_MODE on error.
  - 1460 c. If the MSB of S2 -> et is TPM\_ET\_XOR
    - 1461 i. Use MGF1 to create string X1 of length sealedDataSize. The inputs to MGF1
    - 1462 are; authLastnonceEven, nonceOdd, "XOR", and authHandle -> sharedSecret. The
    - 1463 four concatenated values form the Z value that is the seed for MFG1.
    - 1464 ii. Create o1 by XOR of d1 -> data and X1
  - 1465 d. Else

352

353

1466           i. Create o1 by encrypting d1 -> data using the algorithm indicated by inData ->  
1467           et

1468           ii. Key is from authHandle -> sharedSecret

1469           iii. IV is SHA-1 of (authLastNonceEven || nonceOdd)

1470           e. Set continueAuthSession to FALSE

1471 10.else

1472    a. Set o1 to d1 -> data

1473 11.Set the return secret as o1

1474 12.Return TPM\_SUCCESS

### 147510.3 TPM\_UnBind

#### 1476Start of informative comment:

1477TPM\_UnBind takes the data blob that is the result of a Tspi\_Data\_Bind command and  
1478decrypts it for export to the User. The caller must authorize the use of the key that will  
1479decrypt the incoming blob.

1480TPM\_UnBind operates on a block-by-block basis, and has no notion of any relation between  
1481one block and another.

#### 1482End of informative comment.

#### 1483Incoming 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.



## 1484 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.

## 1485 Description

1486 TPM\_UnBind SHALL operate on a single block only.

## 1487 Actions

1488 The TPM SHALL perform the following:

1489 1. If the inDataSize is 0 the TPM returns TPM\_BAD\_PARAMETER

1490 2. Validate the AuthData to use the key pointed to by keyHandle

1491 3. If the keyUsage field of the key referenced by keyHandle does not have the value  
1492 TPM\_KEY\_BIND or TPM\_KEY\_LEGACY, the TPM must return the error code  
1493 TPM\_INVALID\_KEYUSAGE

1494 4. Decrypt the inData using the key pointed to by keyHandle

1495 5. if (keyHandle -> encScheme does not equal TPM\_ES\_RSAESOAEP\_SHA1\_MGF1) and  
1496 (keyHandle -> keyUsage equals TPM\_KEY\_LEGACY),

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

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

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

1503 6. else

1504 a. Interpret the decrypted data under the assumption that it is a TPM\_BOUND\_DATA  
1505 structure, and validate that the payload type is TPM\_PT\_BIND

1506 b. Set the output parameter outData to the value of TPM\_BOUND\_DATA ->  
1507 payloadData. (Other parameters of TPM\_BOUND\_DATA SHALL NOT be returned.  
1508 Padding associated with the encryption wrapping of inData SHALL NOT be returned.)

- 1509 c. Set the output parameter outDataSize to the size of outData, as deduced from the  
1510 decryption process and the interpretation of TPM\_BOUND\_DATA.
15117. Return the output parameters.

## 151210.4 TPM\_CreateWrapKey

### 1513Start of informative comment:

1514The TPM\_CreateWrapKey command both generates and creates a secure storage bundle for  
1515asymmetric keys.

1516The newly created key can be locked to a specific PCR value by specifying a set of PCR  
1517registers.

1518The key blob does not have a protected identifier. On a platform that does not prevent  
1519unauthorized access to data, a key blob can be exchanged by a lower layer without  
1520detection. The upper layer software must take additional measures to protect the relation  
1521between its identifier of the key blob and the blob itself.

### 1522End of informative comment.

### 1523Incoming Operands and Sizes

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

### 1524Outgoing Operands and Sizes

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

7	20			TPM_AUTHDATA	resAuth	Authorization HMAC key: parentKey.usageAuth.

## 1525 Actions

1526 The TPM SHALL do the following:

1527 1. Validate the AuthData to use the key pointed to by parentHandle. Return  
1528 TPM\_AUTHFAIL on any error.

1529 2. Validate the session type for parentHandle is OSAP.

1530 3. If the TPM is not designed to create a key of the type requested in keyInfo, return the  
1531 error code TPM\_BAD\_KEY\_PROPERTY

1532 4. Verify that parentHandle->keyUsage equals TPM\_KEY\_STORAGE

1533 5. If parentHandle -> keyFlags -> migratable is TRUE and keyInfo -> keyFlags -> migratable  
1534 is FALSE then return TPM\_INVALID\_KEYUSAGE

1535 6. Validate key parameters

1536 a. keyInfo -> keyUsage MUST NOT be TPM\_KEY\_IDENTITY or  
1537 TPM\_KEY\_AUTHCHANGE. If it is, return TPM\_INVALID\_KEYUSAGE

1538 b. If keyInfo -> keyFlags -> migrateAuthority is TRUE then return  
1539 TPM\_INVALID\_KEYUSAGE

1540 7. If TPM\_PERMANENT\_FLAGS -> FIPS is TRUE then

1541 a. If keyInfo -> keySize is less than 1024 return TPM\_NOTFIPS

1542 b. If keyInfo -> authDataUsage specifies TPM\_AUTH\_NEVER return TPM\_NOTFIPS

1543 c. If keyInfo -> keyUsage specifies TPM\_KEY\_LEGACY return TPM\_NOTFIPS

1544 8. If keyInfo -> keyUsage equals TPM\_KEY\_STORAGE or TPM\_KEY\_MIGRATE

1545 i. algorithmID MUST be TPM\_ALG\_RSA

1546 ii. encScheme MUST be TPM\_ES\_RSAESOAEP\_SHA1\_MGF1

1547 iii. sigScheme MUST be TPM\_SS\_NONE

1548 iv. key size MUST be 2048

1549 v. exponentSize MUST be 0

1550 9. Determine the version of key

1551 a. If keyInfo -> ver is 1.1

1552 i. Set V1 to 1

1553 ii. Map wrappedKey to a TPM\_KEY structure

1554 iii. Validate all remaining TPM\_KEY structures

1555 b. Else if keyInfo -> tag is TPM\_TAG\_KEY12

1556 i. Set V1 to 2

1557 ii. Map wrappedKey to a TPM\_KEY12 structure

1558 iii. Validate all remaining TPM\_KEY12 structures

155910. Create DU1 by decrypting dataUsageAuth according to the ADIP indicated by  
1560 authHandle

156111. Create DM1 by decrypting dataMigrationAuth according to the ADIP indicated by  
1562 authHandle

156312. Set continueAuthSession to FALSE

156413. Generate asymmetric key according to algorithm information in keyInfo

156514. Fill in the wrappedKey structure with information from the newly generated key.

1566 a. Set wrappedKey -> encData -> usageAuth to DU1

1567 b. If the KeyFlags -> migratable bit is set to 1, the wrappedKey -> encData ->  
1568 migrationAuth SHALL contain the decrypted value from dataMigrationAuth.

1569 c. If the KeyFlags -> migratable bit is set to 0, the wrappedKey -> encData ->  
1570 migrationAuth SHALL be set to the value tpmProof

157115. If keyInfo->PCRInfoSize is non-zero

1572 a. If V1 is 1

1573 i. Set wrappedKey -> pcrInfo to a TPM\_PCR\_INFO structure using the  
1574 pcrSelection to indicate the PCR's in use

1575 b. Else

1576 i. Set wrappedKey -> pcrInfo to a TPM\_PCR\_INFO\_LONG structure

1577 c. Set wrappedKey -> pcrInfo to keyInfo -> pcrInfo

1578 d. Set wrappedKey -> digestAtCreation to the TPM\_COMPOSITE\_HASH indicated by  
1579 creationPCRSelection

1580 e. If V1 is 2 set wrappedKey -> localityAtCreation to TPM\_STANY\_DATA -> locality

158116. Encrypt the private portions of the wrappedKey structure using the key in parentHandle

158217. Return the newly generated key in the wrappedKey parameter

## 1583 **10.5 TPM\_LoadKey2**

### 1584 **Start of informative comment:**

1585 Before the TPM can use a key to either wrap, unwrap, unbind, seal, unseal, sign or perform  
1586 any other action, it needs to be present in the TPM. The TPM\_LoadKey2 function loads the  
1587 key into the TPM for further use.

1588 The TPM assigns the key handle. The TPM always locates a loaded key by use of the handle.  
1589 The assumption is that the handle may change due to key management operations. It is the  
1590 responsibility of upper level software to maintain the mapping between handle and any  
1591 label used by external software.

1592 To permit this mapping between handle and upper software labels (called key handle  
1593 virtualization), the key handle returned by TPM\_LoadKey2 must not be included in the  
1594 response HMAC. This may cause problems if several keys are authorized using the same  
1595 authorization data. Care should be taken to assign different authorization data to each key.

1596 This command has the responsibility of enforcing restrictions on the use of keys. For  
1597 example, when attempting to load a STORAGE key it will be checked for the restrictions on  
1598 a storage key (2048 size etc.).

1599 The load command must maintain a record of whether any previous key in the key  
1600 hierarchy was bound to a PCR using parentPCRStatus.

1601 The flag parentPCRStatus enables the possibility of checking that a platform passed  
1602 through some particular state or states before finishing in the current state. A grandparent  
1603 key could be linked to state-1, a parent key could be linked to state-2, and a child key could be  
1604 linked to state-3, for example. The use of the child key then indicates that the platform  
1605 passed through states 1 and 2 and is currently in state 3, in this example. TPM\_Startup  
1606 with stType == TPM\_ST\_CLEAR indicates that the platform has been reset, so the platform  
1607 has not passed through the previous states. Hence keys with parentPCRStatus == TRUE  
1608 must be unloaded if TPM\_Startup is issued with stType == TPM\_ST\_CLEAR.

1609 If a TPM\_KEY structure has been decrypted AND the integrity test using "pubDataDigest"  
1610 has passed AND the key is non-migratory, the key must have been created by the TPM. So  
1611 there is every reason to believe that the key poses no security threat to the TPM. While there  
1612 is no known attack from a rogue migratory key, there is a desire to verify that a loaded  
1613 migratory key is a real key, arising from a general sense of unease about execution of  
1614 arbitrary data as a key. Ideally a consistency check would consist of an encrypt/decrypt  
1615 cycle, but this may be expensive. For RSA keys, it is therefore suggested that the  
1616 consistency test consists of dividing the supposed RSA product by the supposed RSA prime,  
1617 and checking that there is no remainder.

### 1618 **End of informative comment.**

## 1619Incoming 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.

## 1620Outgoing 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.

## 1621Actions

1622The TPM SHALL perform the following steps:

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

16252. If parentHandle -> keyUsage is NOT TPM\_KEY\_STORAGE return  
1626 TPM\_INVALID\_KEYUSAGE

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

16294. The TPM MUST handle both TPM\_KEY and TPM\_KEY12 structures

16305. Decrypt the inKey -> privkey to obtain TPM\_STORE\_ASYMKEY structure using the key  
1631 in parentHandle



397  
398

16326. Validate the integrity of inKey and decrypted TPM\_STORE\_ASYMKEY

1633 a. Reproduce inKey -> TPM\_STORE\_ASYMKEY -> pubDataDigest using the fields of  
1634 inKey, and check that the reproduced value is the same as pubDataDigest

16357. Validate the consistency of the key and its key usage.

1636 a. If inKey -> keyFlags -> migratable is TRUE, the TPM SHALL verify consistency of the  
1637 public and private components of the asymmetric key pair. If inKey -> keyFlags ->  
1638 migratable is FALSE, the TPM MAY verify consistency of the public and private  
1639 components of the asymmetric key pair. The consistency of an RSA key pair MAY be  
1640 verified by dividing the supposed (P\*Q) product by a supposed prime and checking that  
1641 there is no remainder.

1642 b. If inKey -> keyUsage is TPM\_KEY\_IDENTITY, verify that inKey->keyFlags->migratable  
1643 is FALSE. If it is not, return TPM\_INVALID\_KEYUSAGE

1644 c. If inKey -> keyUsage is TPM\_KEY\_AUTHCHANGE, return TPM\_INVALID\_KEYUSAGE

1645 d. If inKey -> keyFlags -> migratable equals 0 then verify that TPM\_STORE\_ASYMKEY  
1646 -> migrationAuth equals TPM\_PERMANENT\_DATA -> tpmProof

1647 e. Validate the mix of encryption and signature schemes

1648 f. If TPM\_PERMANENT\_FLAGS -> FIPS is TRUE then

1649 i. If keyInfo -> keySize is less than 1024 return TPM\_NOTFIPS

1650 ii. If keyInfo -> authDataUsage specifies TPM\_AUTH\_NEVER return  
1651 TPM\_NOTFIPS

1652 iii. If keyInfo -> keyUsage specifies TPM\_KEY\_LEGACY return TPM\_NOTFIPS

1653 g. If inKey -> keyUsage is TPM\_KEY\_STORAGE or TPM\_KEY\_MIGRATE

1654 i. algorithmID MUST be TPM\_ALG\_RSA

1655 ii. Key size MUST be 2048

1656 iii. exponentSize MUST be 0

1657 iv. sigScheme MUST be TPM\_SS\_NONE

1658 h. If inKey -> keyUsage is TPM\_KEY\_IDENTITY

1659 i. algorithmID MUST be TPM\_ALG\_RSA

1660 ii. Key size MUST be 2048

1661 iii. exponentSize MUST be 0

1662 iv. encScheme MUST be TPM\_ES\_NONE

1663 i. If the decrypted inKey -> pcrInfo is NULL,

1664 i. The TPM MUST set the internal indicator to indicate that the key is not using  
1665 any PCR registers.

1666 j. Else

1667 i. The TPM MUST store pcrInfo in a manner that allows the TPM to calculate a  
1668 composite hash whenever the key will be in use

1669           ii. The TPM MUST handle both version 1.1 TPM\_PCR\_INFO and 1.2  
1670           TPM\_PCR\_INFO\_LONG structures according to the type of TPM\_KEY structure

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

16748. Perform any processing necessary to make TPM\_STORE\_ASYMKEY key available for  
1675   operations

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

167810. Assign inKeyHandle according to internal TPM rules.

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

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

**1682 10.6 TPM\_GetPubKey****1683Start of informative comment:**

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

**1687End of informative comment.****1688Incoming Operands and Sizes**

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

**1689Outgoing Operands and Sizes**

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

**1690Actions**

1691The TPM SHALL perform the following steps:

16921. If tag = TPM\_TAG\_RQU\_AUTH1\_COMMAND then

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

1694 TPM\_AUTHFAIL

16952. Else

1696 a. Verify that keyHandle -> authDataUsage is TPM\_NO\_READ\_PUBKEY\_AUTH or  
1697 TPM\_AUTH\_NEVER, on error return TPM\_AUTHFAIL

16983. If keyHandle == TPM\_KH\_SRK then

1699 a. If TPM\_PERMANENT\_FLAGS -> readSRKPub is FALSE then return  
1700 TPM\_INVALID\_KEYHANDLE

17014. If keyHandle -> pcrInfoSize is not 0

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

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

1705 ii. If specified validate any locality requests

17065. Create a TPM\_PUBKEY structure and return

**170710.7 TPM\_Sealx****1708Start of informative comment:**

1709The TPM\_Sealx command works exactly like the TPM\_Seal command with the additional  
1710requirement of encryption for the inData parameter. This command also places in the  
1711sealed blob the information that the TPM\_Unseal also requires encryption.

1712TPM\_Sealx requires the use of 1.2 data structures. The actions are the same as TPM\_Seal  
1713without the checks for 1.1 data structure usage.

1714The method of incrementing the symmetric key counter value is different from that used by  
1715some standard crypto libraries (e.g. openssl, Java JCE) that increment the entire counter  
1716value. TPM users should be aware of this to avoid errors when the counter wraps.

**1717End of informative comment.****1718Incoming 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.

## 1719Outgoing 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.

## 1720Actions

17211. Validate the authorization to use the key pointed to by keyHandle
17222. If the inDataSize is 0 the TPM returns TPM\_BAD\_PARAMETER
17233. If the keyUsage field of the key indicated by keyHandle does not have the value 1724 TPM\_KEY\_STORAGE, the TPM must return the error code TPM\_INVALID\_KEYUSAGE.
17254. If the keyHandle points to a migratable key then the TPM MUST return the error code 1726 TPM\_INVALID\_KEY\_USAGE.
17275. Create S1 a TPM\_STORED\_DATA12 structure
17286. Set S1 -> encDataSize to 0
17297. Set S1 -> encData to all zeros
17308. Set S1 -> sealInfoSize to pcrInfoSize
17319. If pcrInfoSize is not 0 then
- 1732 a. Validate pcrInfo as a valid TPM\_PCR\_INFO\_LONG structure, return TPM\_BADINDEX
  - 1733 on error
  - 1734 b. Set S1 -> sealInfo -> creationPCRSelection to pcrInfo -> creationPCRSelection
  - 1735 c. Set S1 -> sealInfo -> releasePCRSelection to pcrInfo -> releasePCRSelection
  - 1736 d. Set S1 -> sealInfo -> digestAtRelease to pcrInfo -> digestAtRelease
  - 1737 e. Set S1 -> sealInfo -> localityAtRelease to pcrInfo -> localityAtRelease
  - 1738 f. Create h2 the composite hash of the TPM\_STCLEAR\_DATA -> PCR selected by
  - 1739 pcrInfo -> creationPCRSelection
  - 1740 g. Set S1 -> sealInfo -> digestAtCreation to h2
  - 1741 h. Set S1 -> sealInfo -> localityAtCreation to TPM\_STANY\_DATA -> localityModifier
174210. Create S2 a TPM\_SEALED\_DATA structure

424

425

1743 11. Create a1 by decrypting encAuth according to the ADIP indicated by authHandle.

1744     a. If authHandle indicates XOR encryption for the AuthData secrets

1745         i. Set S1 -> et to TPM\_ET\_XOR || TPM\_ET\_KEY

1746             (1) TPM\_ET\_KEY is added because TPM\_Unseal uses zero as a special value  
1747             indicating no encryption.

1748     b. Else

1749         i. Set S1 -> et to the algorithm indicated by authHandle

1750 12. The TPM provides NO validation of a1. Well-known values (like all zeros) are valid and  
1751     possible.

1752 13. If authHandle indicates XOR encryption

1753     a. Use MGF1 to create string X2 of length inDataSize. The inputs to MGF1 are;  
1754     authLastNonceEven, nonceOdd, "XOR", and authHandle -> sharedSecret. The four  
1755     concatenated values form the Z value that is the seed for MFG1.

1756     b. Create o1 by XOR of inData and X2

1757 14. Else

1758     a. Create o1 by decrypting inData using the algorithm indicated by authHandle

1759     b. Key is from authHandle -> sharedSecret

1760     c. CTR is SHA-1 of (authLastNonceEven || nonceOdd)

1761 15. Create S2 a TPM\_SEALED\_DATA structure

1762     a. Set S2 -> payload to TPM\_PT\_SEAL

1763     b. Set S2 -> tpmProof to TPM\_PERMANENT\_DATA -> tpmProof

1764     c. Create h3 the SHA-1 of S1

1765     d. Set S2 -> storedDigest to h3

1766     e. Set S2 -> authData to a1

1767     f. Set S2 -> dataSize to inDataSize

1768     g. Set S2 -> data to o1

1769 16. Validate that the size of S2 can be encrypted by the key pointed to by keyHandle, return  
1770     TPM\_BAD\_DATASIZE on error

1771 17. Create s3 the encryption of S2 using the key pointed to by keyHandle

1772 18. Set continueAuthSession to FALSE

1773 19. Set S1 -> encDataSize to the size of s3

1774 20. Set S1 -> encData to s3

1775 21. Return S1 as sealedData

## 1776**11. Migration**

### 1777**Start of informative comment:**

1778The migration of a key from one TPM to another is a vital aspect to many use models of the  
1779TPM. The migration commands are the commands that allow this operation to occur.

1780There are two types of migratable keys, the version 1.1 migratable keys and the version 1.2  
1781certifiable migratable keys.

### 1782**End of informative comment.**

## 1783**11.1 TPM\_CreateMigrationBlob**

### 1784**Start of informative comment:**

1785The TPM\_CreateMigrationBlob command implements the first step in the process of moving  
1786a migratable key to a new parent or platform. Execution of this command requires  
1787knowledge of the migrationAuth field of the key to be migrated.

1788Migrate mode is generally used to migrate keys from one TPM to another for backup,  
1789upgrade or to clone a key on another platform. To do this, the TPM needs to create a data  
1790blob that another TPM can deal with. This is done by loading in a backup public key that  
1791will be used by the TPM to create a new data blob for a migratable key.

1792The TPM Owner does the selection and authorization of migration public keys at any time  
1793prior to the execution of TPM\_CreateMigrationBlob by performing the  
1794TPM\_AuthorizeMigrationKey command.

1795IReWrap mode is used directly to move the key to a new parent (on either this platform or  
1796another). The TPM simply re-encrypts the key using a new parent, and outputs a normal  
1797encrypted element that can be subsequently used by a TPM\_LoadKey command.

1798TPM\_CreateMigrationBlob implicitly cannot be used to migrate a non-migratory key. No  
1799explicit check is required. Only the TPM knows tpmProof. Therefore, it is impossible for the  
1800caller to submit an AuthData value equal to tpmProof and migrate a non-migratory key.

### 1801**End of informative comment.**



## 1802 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.

1803

## 1804Outgoing Operands and Sizes

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

## 1805Description

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

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

## 1809Actions

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

18112. Validate that parentHandle -> keyUsage is TPM\_KEY\_STORAGE, if not return  
1812 TPM\_INVALID\_KEYUSAGE

18133. Create d1 a TPM\_STORE\_ASYMKEY structure by decrypting encData using the key  
1814 pointed to by parentHandle.

1815 a. Verify that d1 -> payload is TPM\_PT\_ASYM.

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

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

18216. If migrationType == TPM\_MS\_MIGRATE the TPM SHALL perform the following actions:

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

441  
442  
443

1823           i. K1 = d1.privKey[0..19] (d1.privKey.keyLength + 16 bytes of d1.privKey.key),  
1824           sizeof(K1) = 20

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

1826   b. Build M1 a TPM\_MIGRATE\_ASYMKEY structure

1827           i. TPM\_MIGRATE\_ASYMKEY.payload = TPM\_PT\_MIGRATE

1828           ii. TPM\_MIGRATE\_ASYMKEY.usageAuth = d1.usageAuth

1829           iii. TPM\_MIGRATE\_ASYMKEY.pubDataDigest = d1. pubDataDigest

1830           iv. TPM\_MIGRATE\_ASYMKEY.partPrivKeyLen = 112 – 127.

1831           v. TPM\_MIGRATE\_ASYMKEY.partPrivKey = K2

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

1834           i. m = M1 the TPM\_MIGRATE\_ASYMKEY structure

1835           ii. pHash = d1->migrationAuth

1836           iii. seed = s1 = K1

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

1839   e. Create x1 by XOR of o1 with r1

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

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

18427. If migrationType == TPM\_MS\_REWRAP the TPM SHALL perform the following actions:

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

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

18468. Else

1847   a. Return TPM\_BAD\_PARAMETER

## 1848**11.2 TPM\_ConvertMigrationBlob**

### 1849**Start of informative comment:**

1850This command takes a migration blob and creates a normal wrapped blob. The migrated  
1851blob must be loaded into the TPM using the normal TPM\_LoadKey function.

1852Note that the command migrates private keys, only. The migration of the associated public  
1853keys is not specified by TPM because they are not security sensitive. Migration of the  
1854associated public keys may be specified in a platform specific specification. A TPM\_KEY  
1855structure must be recreated before the migrated key can be used by the target TPM in a  
1856TPM\_LoadKey command.

### 1857**End of informative comment.**

### 1858**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

1859

**1860 Outgoing Operands and Sizes**

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

**1861 Action**

1862 The TPM SHALL perform the following:

18631. Validate the AuthData to use the key in parentHandle
18642. If the keyUsage field of the key referenced by parentHandle does not have the value  
1865 TPM\_KEY\_STORAGE, the TPM must return the error code TPM\_INVALID\_KEYUSAGE
18663. Create d1 by decrypting the inData area using the key in parentHandle
18674. Create o1 by XOR d1 and random parameter
18685. Create m1 a TPM\_MIGRATE\_ASYMKEY structure, seed and pHash by OAEP decoding o1
18696. Create k1 by combining seed and the TPM\_MIGRATE\_ASYMKEY -> partPrivKey field
18707. Create d2 a TPM\_STORE\_ASYMKEY structure
  - 1871 a. Verify that m1 -> payload == TPM\_PT\_MIGRATE
  - 1872 b. Set d2 -> payload = TPM\_PT\_ASYM
  - 1873 c. Set d2 -> usageAuth to m1 -> usageAuth
  - 1874 d. Set d2 -> migrationAuth to pHash
  - 1875 e. Set d2 -> pubDataDigest to m1 -> pubDataDigest
  - 1876 f. Set d2 -> privKey field to k1
18778. Create outData using the key in parentHandle to perform the encryption

## 1878**11.3 TPM\_AuthorizeMigrationKey**

### 1879**Start of informative comment:**

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

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

### 1885**End of informative comment.**

### 1886**Incoming Operands and Sizes**

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

### 1887**Outgoing Operands and Sizes**

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

1888

### 1889**Action**

1890The TPM SHALL perform the following:

460

461

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

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

18953. Create a f1 a TPM\_MIGRATIONKEYAUTH structure

18964. Verify that migrationKey-> algorithmParms -> encScheme is  
1897 TPM\_ES\_RSAESOAEP\_SHA1\_MGF1, and return the error code  
1898 TPM\_INAPPROPRIATE\_ENC if it is not

18995. Set f1 -> migrationKey to the input migrationKey

19006. Set f1 -> migrationScheme to the input migrationScheme

19017. Create v1 by concatenating (migrationKey || migrationScheme ||  
1902 TPM\_PERMANENT\_DATA -> tpmProof)

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

19049. Set f1 -> digest to h1

190510. Return f1 as outData

## 1906**11.4 TPM\_MigrateKey**

### 1907**Start of informative comment:**

1908The TPM\_MigrateKey command performs the function of a migration authority.

1909The command is relatively simple; it just decrypts the input packet (coming from  
1910TPM\_CreateMigrationBlob or TPM\_CMK\_CreateBlob) and then re-encrypts it with the input  
1911public key. The output of this command would then be sent to TPM\_ConvertMigrationBlob  
1912or TPM\_CMK\_ConvertMigration on the target TPM.

1913TPM\_MigrateKey does not make ANY assumptions about the contents of the encrypted blob.  
1914Since it does not have the XOR string, it cannot actually determine much about the key  
1915that is being migrated.

1916This command exists to permit the TPM to be a migration authority. If used in this way, it is  
1917expected that the physical security of the system containing the TPM and the AuthData  
1918value for the MA key would be tightly controlled.

1919To prevent the execution of this command using any other key as a parent key, this  
1920command works only if keyUsage for maKeyHandle is TPM\_KEY\_MIGRATE.

### 1921**End of informative comment.**

### 1922**Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_MigrateKey
4	4			TPM_KEY_HANDLE	maKeyHandle	Handle of the key to be used to migrate the key.
5	<>	2S	<>	TPM_PUBKEY	pubKey	Public key to which the blob is to be migrated
6	4	3S	4	UINT32	inDataSize	The size of inData
7	<>	4S	<>	BYTE[]	inData	The input blob
8	4			TPM_AUTHHANDLE	maAuthHandle	The authorization session handle used for maKeyHandle.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with certAuthHandle
10	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	keyAuth	The authorization session digest for the inputs and key to be signed. HMAC key: maKeyHandle.usageAuth.



## 1923 Outgoing Operands and Sizes

Param		HMAC		Type	Name	Description
#	Sz	#	Sz			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_MigrateKey
4	4	3S	4	UINT32	outDataSize	The used size of the output area for outData
5	<>	4S	<>	BYTE[]	outData	The re-encrypted blob
6	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with certAuthHandle
7	1	4H1	1	BOOL	continueAuthSession	Continue use flag for cert key session
8	20			TPM_AUTHDATA	keyAuth	The authorization session digest for the target key. HMAC key: maKeyHandle.usageAuth

## 1924 Actions

19251. Validate that keyAuth authorizes the use of the key pointed to by maKeyHandle
19262. The TPM validates that the key pointed to by maKeyHandle has a key usage value of  
1927 TPM\_KEY\_MIGRATE, and that the allowed encryption scheme is  
1928 TPM\_ES\_RSAESOAEP\_SHA1\_MGF1.
19293. The TPM validates that pubKey is of a size supported by the TPM and that its size is  
1930 consistent with the input blob and maKeyHandle.
19314. The TPM decrypts inData and re-encrypts it using pubKey.

## 193211.5 TPM\_CMK\_SetRestrictions

### 1933Start of informative comment:

1934This command is used by the Owner to dictate the usage of a certified-migration key with  
1935delegated authorization (authorization other than actual owner authorization).

1936This command is provided for privacy reasons and must not itself be delegated, because a  
1937certified-migration-key may involve a contractual relationship between the Owner and an  
1938external entity.

1939Since restrictions are validated at DSAP session use, there is no need to invalidate DSAP  
1940sessions when the restriction value changes.

### 1941End of informative comment.

### 1942Incoming 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

### 1943Outgoing 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.

### 1944Description

1945TPM\_PERMANENT\_DATA -> restrictDelegate is used as follows

19461. If the session type is TPM\_PID\_DSAP and TPM\_KEY -> keyFlags -> migrateAuthority is  
1947 TRUE

1948 a. If

1949TPM\_KEY\_USAGE is TPM\_KEY\_SIGNING and restrictDelegate ->  
1950TPM\_CMK\_DELEGATE\_SIGNING is TRUE, or

1951TPM\_KEY\_USAGE is TPM\_KEY\_STORAGE and restrictDelegate ->  
1952TPM\_CMK\_DELEGATE\_STORAGE is TRUE, or

1953TPM\_KEY\_USAGE is TPM\_KEY\_BIND and restrictDelegate -> TPM\_CMK\_DELEGATE\_BIND  
1954is TRUE, or

1955TPM\_KEY\_USAGE is TPM\_KEY\_LEGACY and restrictDelegate ->  
1956TPM\_CMK\_DELEGATE\_LEGACY is TRUE, or

1957TPM\_KEY\_USAGE is TPM\_KEY\_MIGRATE and restrictDelegate ->  
1958TPM\_CMK\_DELEGATE\_MIGRATE is TRUE

1959then the key can be used.

1960 b. Else return TPM\_INVALID\_KEYUSAGE.

### 1961**Actions**

19621. Validate the ordinal and parameters using TPM Owner authentication, return  
1963 TPM\_AUTHFAIL on error

19642. Set TPM\_PERMANENT\_DATA -> TPM\_CMK\_DELEGATE -> restrictDelegate = restriction

19653. Return TPM\_SUCCESS

## 1966**11.6 TPM\_CMK\_ApproveMA**

### 1967**Start of informative comment:**

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

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

### 1973**End of informative comment.**

### 1974**Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_ApproveMA
4	20	2S	20	TPM_DIGEST	migrationAuthorityDigest	A digest of a TPM_MSA_COMPOSITE structure (itself one or more digests of public keys belonging to migration authorities)
5	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
6	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
7	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
8	20			TPM_AUTHDATA	ownerAuth	Authorization HMAC, key: ownerAuth.

### 1975**Outgoing Operands and Sizes**

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

### 1976**Action**

1977The TPM SHALL perform the following:

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

19792. Create M2 a TPM\_CMK\_MA\_APPROVAL structure

1980 a. Set M2 ->migrationAuthorityDigest to migrationAuthorityDigest

19813. Set outData = HMAC(M2) using tpmProof as the secret

19824. Return TPM\_SUCCESS

## 198311.7 TPM\_CMK\_CreateKey

### 1984Start of informative comment:

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

1987TPM\_CMK\_CreateKey is very similar to TPM\_CreateWrapKey, but: (1) the resultant key must  
1988be a migratable key and can be migrated only by TPM\_CMK\_CreateBlob; (2) the command is  
1989Owner authorized via a ticket.

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

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

### 1997End of informative comment.

### 1998Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_CreateKey
4	4			TPM_KEY_HANDLE	parentHandle	Handle of a loaded key that can perform key wrapping.
5	20	2S	20	TPM_ENCAUTH	dataUsageAuth	Encrypted usage AuthData for the key.
6	<>	3S	<>	TPM_KEY12	keyInfo	Information about key to be created, pubkey.keyLength and keyInfo.encData elements are 0. MUST be TPM_KEY12
7	20	4S	20	TPM_HMAC	migrationAuthorityApproval	A ticket, created by the TPM Owner using TPM_CMK_ApproveMA, approving a TPM_MSA_COMPOSITE structure
8	20	5S	20	TPM_DIGEST	migrationAuthorityDigest	The digest of a TPM_MSA_COMPOSITE structure
9	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for parent key authorization. Must be an OSAP session.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
10	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
11	1	4H1	1	BOOL	continueAuthSession	Ignored
12	20			TPM_AUTHDATA	pubAuth	The authorization session digest that authorizes the use of the public key in parentHandle. HMAC key: parentKey.usageAuth.

## 1999Outgoing 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.

## 2000Actions

2001The TPM SHALL do the following:

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

20042. Validate the session type for parentHandle is OSAP

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

20074. Verify that parentHandle->keyUsage equals TPM\_KEY\_STORAGE

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

20096. If keyInfo -> keyFlags -> migratable is FALSE, return TPM\_INVALID\_KEYUSAGE

20107. If keyInfo -> keyFlags -> migrateAuthority is FALSE , return TPM\_INVALID\_KEYUSAGE

20118. Verify that the migration authority is authorized

2012 a. Create M1 a TPM\_CMK\_MA\_APPROVAL structure

2013 i. Set M1 ->migrationAuthorityDigest to migrationAuthorityDigest

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

20169. Validate key parameters

2017 a. keyInfo -> keyUsage MUST NOT be TPM\_KEY\_IDENTITY or  
2018 TPM\_KEY\_AUTHCHANGE. If it is, return TPM\_INVALID\_KEYUSAGE

201910.If TPM\_PERMANENT\_FLAGS -> FIPS is TRUE then

2020 a. If keyInfo -> keySize is less than 1024 return TPM\_NOTFIPS

2021 b. If keyInfo -> authDataUsage specifies TPM\_AUTH\_NEVER return TPM\_NOTFIPS

2022 c. If keyInfo -> keyUsage specifies TPM\_KEY\_LEGACY return TPM\_NOTFIPS

- 202311.If keyInfo -> keyUsage equals TPM\_KEY\_STORAGE or TPM\_KEY\_MIGRATE
- 2024 a. algorithmID MUST be TPM\_ALG\_RSA
- 2025 b. encScheme MUST be TPM\_ES\_RSAESOAEP\_SHA1\_MGF1
- 2026 c. sigScheme MUST be TPM\_SS\_NONE
- 2027 d. key size MUST be 2048
- 2028 e. exponentSize MUST be 0
- 202912.If keyInfo -> tag is NOT TPM\_TAG\_KEY12 return error TPM\_INVALID\_STRUCTURE
- 203013.Map wrappedKey to a TPM\_KEY12 structure
- 203114.Create DU1 by decrypting dataUsageAuth according to the ADIP indicated by  
2032 authHandle.
- 203315.Set continueAuthSession to FALSE
- 203416.Generate asymmetric key according to algorithm information in keyInfo
- 203517.Fill in the wrappedKey structure with information from the newly generated key.
- 2036 a. Set wrappedKey -> encData -> usageAuth to DU1
- 2037 b. Set wrappedKey -> encData -> payload to TPM\_PT\_MIGRATE\_RESTRICTED
- 2038 c. Create thisPubKey, a TPM\_PUBKEY structure containing wrappedKey's public key  
2039 and algorithm parameters
- 2040 d. Create M2 a TPM\_CMK\_MIGAUTH structure
- 2041 i. Set M2 -> msaDigest to migrationAuthorityDigest
- 2042 ii. Set M2 -> pubKeyDigest to SHA-1 (thisPubKey)
- 2043 e. Set wrappedKey -> encData -> migrationAuth equal to HMAC(M2), using tpmProof as  
2044 the shared secret
- 204518.If keyInfo->PCRInfoSize is non-zero
- 2046 a. Set wrappedKey -> pcrInfo to a TPM\_PCR\_INFO\_LONG structure
- 2047 b. Set wrappedKey -> pcrInfo to keyInfo -> pcrInfo
- 2048 c. Set wrappedKey -> digestAtCreation to the TPM\_COMPOSITE\_HASH indicated by  
2049 creationPCRSelection
- 2050 d. Set wrappedKey -> localityAtCreation to TPM\_STANY\_FLAGS -> localityModifier
- 205119.Encrypt the private portions of the wrappedKey structure using the key in parentHandle
- 205220.Return the newly generated key in the wrappedKey parameter



## 205311.8 TPM\_CMK\_CreateTicket

### 2054Start of informative comment:

2055The TPM\_CMK\_CreateTicket command uses a public key to verify the signature over a  
2056digest.

2057TPM\_CMK\_CreateTicket returns a ticket that can be used to prove to the same TPM that  
2058signature verification with a particular public key was successful.

### 2059End of informative comment.

### 2060Incoming 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.

## 2061Outgoing 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.

## 2062Actions

2063The TPM SHALL do the following:

20641. Validate the TPM Owner authentication to use the command

20652. Validate that the key type and algorithm are correct

2066 a. Validate that verificationKey -> algorithmParms -> algorithmID == TPM\_ALG\_RSA

2067 b. Validate that verificationKey -> algorithmParms -> encScheme == TPM\_ES\_NONE

2068 c. Validate that verificationKey -> algorithmParms -> sigScheme is  
2069 TPM\_SS\_RSASSAPKCS1v15\_SHA1 or TPM\_SS\_RSASSAPKCS1v15\_INFO

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

20724. Create M2 a TPM\_CMK\_SIGTICKET

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

2074 b. Set M2 -> signedData to signedData

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

20766. Return TPM\_SUCCESS

## 2077 **11.9 TPM\_CMK\_CreateBlob**

### 2078 **Start of informative comment:**

2079 TPM\_CMK\_CreateBlob command is very similar to TPM\_CreateMigrationBlob, except that it:  
2080 (1) uses an extra ticket (restrictedKeyAuth) instead of a migrationAuth authorization  
2081 session; (2) uses the migration options TPM\_MS\_RESTRICT\_MIGRATE or  
2082 TPM\_MS\_RESTRICT\_APPROVE; (3) produces a wrapped key blob whose migrationAuth is  
2083 independent of tpmProof.

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

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

### 2094 **End of informative comment.**

## 2095Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CMK_CreateBlob
4	4			TPM_KEY_HANDLE	parentHandle	Handle of the parent key that can decrypt encData.
5	2	2S	2	TPM_MIGRATE_SCHEME	migrationType	The migration type, either TPM_MS_RESTRICT_MIGRATE or TPM_MS_RESTRICT_APPROVE
6	<>	3S	<>	TPM_MIGRATIONKEYAUTH	migrationKeyAuth	Migration public key and its authorization session digest.
7	20	4S	20	TPM_DIGEST	pubSourceKeyDigest	The digest of the TPM_PUBKEY of the entity to be migrated
8	4	5S	4	UINT32	msaListSize	The size of the msaList parameter, which is a variable length TPM_MSA_COMPOSITE structure
9	<>	6S	<>	TPM_MSA_COMPOSITE	msaList	One or more digests of public keys belonging to migration authorities
10	4	7S	4	UINT32	restrictTicketSize	The size of the restrictTicket parameter
11	<>	8S	<>	BYTE[]	restrictTicket	If migrationType is TPM_MS_RESTRICT_APPROVE, a TPM_CMK_AUTH structure, containing the digests of the public keys belonging to the Migration Authority, the destination parent key and the key-to-be-migrated.
12	4	9S	4	UINT32	sigTicketSize	The size of the sigTicket parameter
13	<>	10S	<>	BYTE[]	sigTicket	If migrationType is TPM_MS_RESTRICT_APPROVE, a TPM_HMAC structure, generated by the TPM, signaling a valid signature over restrictTicket
14	4	11S	4	UINT32	encDataSize	The size of the encData parameter
15	<>	12S	<>	BYTE[]	encData	The encrypted entity that is to be modified.
16	4			TPM_AUTHHANDLE	parentAuthHandle	The authorization session handle used for the parent key.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
17	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with parentAuthHandle
18	1	4H1	1	BOOL	continueAuthSession	Continue use flag for parent session
19	20		20	TPM_AUTHDATA	parentAuth	HMAC key: parentKey.usageAuth.

## 2096Outgoing Operands and Sizes

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

## 2097Description

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

## 2099Actions

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

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

2103 a. The TPM MAY ignore migrationType.

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

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

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

21076. Verify that d1 -> payload == TPM\_PT\_MIGRATE\_RESTRICTED or  
2108 TPM\_PT\_MIGRATE\_EXTERNAL

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

2110 a. Create M2 a TPM\_CMK\_MIGAUTH structure

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

2112 ii. Set M2 -> pubKeyDigest to pubSourceKeyDigest

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

21158. If migrationKeyAuth -> migrationScheme == TPM\_MS\_RESTRICT\_MIGRATE

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

```

2117         i. For one of n=1 to n=(msaList -> MSAlist), verify that SHA-1[migrationKeyAuth
2118            -> migrationKey] == msaList -> migAuthDigest[n]
2119     b. Validate that the MA key is the correct type
2120         i. Validate that migrationKeyAuth -> migrationKey -> algorithmParms ->
2121            algorithmID == TPM_ALG_RSA
2122         ii. Validate that migrationKeyAuth -> migrationKey -> algorithmParms ->
2123            encScheme is an encryption scheme supported by the TPM
2124         iii. Validate that migrationKeyAuth -> migrationKey -> algorithmParms ->
2125            sigScheme is TPM_SS_NONE
2126     c. The TPM MAY validate that restrictTicketSize is zero.
2127     d. The TPM MAY validate that sigTicketSize is zero.
21289. else If migrationKeyAuth -> migrationScheme == TPM_MS_RESTRICT_APPROVE
2129     a. Verify that the intended migration destination has been approved by the MSA:
2130         i. Verify that for one of the n=1 to n=(msaList -> MSAlist) values of msaList ->
2131            migAuthDigest[n], sigTicket == HMAC (V1) using tpmProof as the secret where V1
2132            is a TPM_CMK_SIGTICKET structure such that:
2133                (1) V1 -> verKeyDigest = msaList -> migAuthDigest[n]
2134                (2) V1 -> signedData = SHA-1[restrictTicket]
2135         ii. If [restrictTicket -> destinationKeyDigest] != SHA-1[migrationKeyAuth ->
2136            migrationKey], return error TPM_MA_DESTINATION
2137         iii. If [restrictTicket -> sourceKeyDigest] != pubSourceKeyDigest, return error
2138            TPM_MA_SOURCE
213910. Else return with error TPM_BAD_PARAMETER.
214011. Build two bytes array, K1 and K2, using d1:
2141     a. K1 = TPM_STORE_ASYMKEY.privKey[0..19]
2142        (TPM_STORE_ASYMKEY.privKey.keyLength + 16 bytes of
2143        TPM_STORE_ASYMKEY.privKey.key), sizeof(K1) = 20
2144     b. K2 = TPM_STORE_ASYMKEY.privKey[20..131] (position 16-127 of
2145        TPM_STORE_ASYMKEY . privKey.key), sizeof(K2) = 112
214612. Build M1 a TPM_MIGRATE_ASYMKEY structure
2147     a. TPM_MIGRATE_ASYMKEY.payload = TPM_PT_CMK_MIGRATE
2148     b. TPM_MIGRATE_ASYMKEY.usageAuth = TPM_STORE_ASYMKEY.usageAuth
2149     c. TPM_MIGRATE_ASYMKEY.pubDataDigest = TPM_STORE_ASYMKEY. pubDataDigest
2150     d. TPM_MIGRATE_ASYMKEY.partPrivKeyLen = 112 - 127.
2151     e. TPM_MIGRATE_ASYMKEY.partPrivKey = K2
215213. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP
2153     encoding of m using OAEP parameters m, pHash, and seed
2154     a. m is the previously created M1

```

532

533

2155     b. pHash = SHA-1( SHA-1[msaList] || pubSourceKeyDigest)

2156     c. seed = s1 = the previously created K1

2157 14. Create r1 a random value from the TPM RNG. The size of r1 MUST be the size of o1.

2158     Return r1 in the random parameter

2159 15. Create x1 by XOR of o1 with r1

2160 16. Copy r1 into the output field “random”

2161 17. Encrypt x1 with the migrationKeyAuth-> migrationKey

## 216211.10 TPM\_CMK\_ConvertMigration

### 2163Start of informative comment:

2164TPM\_CMK\_ConvertMigration completes the migration of certified migration blobs.

2165This command takes a certified migration blob and creates a normal wrapped blob with  
2166payload type TPM\_PT\_MIGRATE\_EXTERNAL. The migrated blob must be loaded into the  
2167TPM using the normal TPM\_LoadKey function.

2168Note that the command migrates private keys, only. The migration of the associated public  
2169keys is not specified by TPM because they are not security sensitive. Migration of the  
2170associated public keys may be specified in a platform specific specification. A TPM\_KEY  
2171structure must be recreated before the migrated key can be used by the target TPM in a  
2172TPM\_LoadKey command.

2173TPM\_CMK\_ConvertMigration checks that one of the MAs implicitly listed in the  
2174migrationAuth of the target key has approved migration of the target key to the destination  
2175(parent) key, and that the settings (flags etc.) in the target key are those of a CMK.

### 2176End of informative comment.

### 2177Incoming 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



## 2178 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

## 2179 Action

21801. Validate the AuthData to use the key in parentHandle
21812. If the keyUsage field of the key referenced by parentHandle does not have the value  
2182 TPM\_KEY\_STORAGE, the TPM must return the error code TPM\_INVALID\_KEYUSAGE
21833. Create d1 by decrypting the migratedKey -> encData area using the key in parentHandle
21844. Create o1 by XOR d1 and random parameter
21855. Create m1 a TPM\_MIGRATE\_ASYMKEY, seed and pHash by OAEP decoding o1
21866. Create migratedPubKey a TPM\_PUBKEY structure corresponding to migratedKey
- 2187 a. Verify that pHash == SHA-1( SHA-1[msaList] || SHA-1(migratedPubKey )
21887. Create k1 by combining seed and the TPM\_MIGRATE\_ASYMKEY -> partPrivKey field
21898. Create d2 a TPM\_STORE\_ASYMKEY structure.
- 2190 a. Set the TPM\_STORE\_ASYMKEY -> privKey field to k1
- 2191 b. Set d2 -> usageAuth to m1 -> usageAuth
- 2192 c. Set d2 -> pubDataDigest to m1 -> pubDataDigest
21939. Verify that parentHandle-> keyFlags -> migratable == FALSE
219410. Verify that m1 -> payload == TPM\_PT\_CMK\_MIGRATE then set d2-> payload =  
2195 TPM\_PT\_MIGRATE\_EXTERNAL
219611. Verify that for one of the n=1 to n=(msaList -> MSaList) values of msaList ->  
2197 migAuthDigest[n] sigTicket == HMAC (V1) using tpmProof as the secret where V1 is a  
2198 TPM\_CMK\_SIGTICKET structure such that:
- 2199 a. V1 -> verKeyDigest = msaList -> migAuthDigest[n]
- 2200 b. V1 -> signedData = SHA-1[restrictTicket]
220112. Create parentPubKey, a TPM\_PUBKEY structure corresponding to parentHandle

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

220414.Verify that migratedKey is corresponding to d2

220515.If migratedKey -> keyFlags -> migratable is FALSE, and return error  
2206 TPM\_INVALID\_KEYUSAGE

220716.If migratedKey -> keyFlags -> migrateAuthority is FALSE, return error  
2208 TPM\_INVALID\_KEYUSAGE

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

221118.Create M2 a TPM\_CMK\_MIGAUTH structure

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

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

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

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

## 2216 **12. Maintenance Functions (optional)**

### 2217 **Start of informative comment:**

2218 When a maintenance archive is created with generateRandom FALSE, the maintenance blob  
2219 is XOR encrypted with the owner authorization before encryption with the maintenance  
2220 public key. This prevents the manufacturer from obtaining plaintext data. The receiving  
2221 TPM must have the same owner authorization as the sending TPM in order to XOR decrypt  
2222 the archive.

2223 When generateRandom is TRUE, the maintenance blob is XOR encrypted with random data,  
2224 which is also returned. This permits someone trusted by the Owner to load the  
2225 maintenance archive into the replacement platform in the absence of the Owner and  
2226 manufacturer, without the Owner having to reveal information about his auth value. The  
2227 receiving and sending TPM's may have different owner authorizations. The random data is  
2228 transferred from the sending TPM owner to the receiving TPM owner out of band, so the  
2229 maintenance blob remains hidden from the manufacturer.

2230 This is a typical maintenance sequence:

2231 1. Manufacturer:

2232 • generates maintenance key pair

2233 • gives public key to TPM1 owner

2234 2. TPM1: TPM\_LoadManuMaintPub

2235 • load maintenance public key

2236 3. TPM1: TPM\_CreateMaintenanceArchive

2237 • XOR encrypt with owner auth or random

2238 • encrypt the maintenance archive with maintenance public key

2239 4. TPM2:

2240 • Take ownership

2241 • Create and activate an AIK

2242 • Certify the SRK with the AIK, proving that the SRK came from a legitimate TPM

2243 5. Manufacturer:

2244 • decrypt maintenance archive with maintenance private key

2245 • (still XOR encrypted with owner auth or random)

2246 • validate the TPM2 SRK certification

2247 • encrypt the maintenance archive with TPM2 SRK public key

2248 6. TPM2: TPM\_LoadMaintenanceArchive

2249 • decrypt the maintenance archive with SRK private key

2250 • XOR decrypt with owner auth or random

2251 **End of informative comment.**

2252

2254**12.1 TPM\_CreateMaintenanceArchive**2255**Start of informative comment:**

2256This command creates the maintenance archive. It can only be executed by the owner, and  
2257may be shut off with the TPM\_KillMaintenanceFeature command.

2258**End of informative comment.**2259**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.

2260**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.

2261**Actions**

2262Upon authorization being confirmed this command does the following:

22631. Validates that the TPM\_PERMANENT\_FLAGS -> allowMaintenance is TRUE. If it is  
2264 FALSE, the TPM SHALL return TPM\_DISABLED\_CMD and exit this capability.
22652. Validates the TPM Owner AuthData.
22663. If the value of TPM\_PERMANENT\_DATA -> manuMaintPub is zero, the TPM MUST  
2267 return the error code TPM\_KEYNOTFOUND
22684. Build a1 a TPM\_KEY structure using the SRK. The encData field is not a normal  
2269 TPM\_STORE\_ASYMKEY structure but rather a TPM\_MIGRATE\_ASYMKEY structure built  
2270 using the following actions.
22715. Build a TPM\_STORE\_PRIVKEY structure from the SRK. This privKey element should be  
2272 132 bytes long for a 2K RSA key.
22736. Create k1 and k2 by splitting the privKey element created in step 4 into 2 parts. k1 is  
2274 the first 20 bytes of privKey, k2 contains the remainder of privKey.
22757. Build m1 by creating and filling in a TPM\_MIGRATE\_ASYMKEY structure
- 2276 a. m1 -> usageAuth is set to TPM\_PERMANENT\_DATA -> tpmProof
- 2277 b. m1 -> pubDataDigest is set to the digest value of the SRK fields from step 4
- 2278 c. m1 -> payload is set to TPM\_PT\_MAINT
- 2279 d. m1 -> partPrivKey is set to k2
22808. Create o1 (which SHALL be 198 bytes for a 2048 bit RSA key) by performing the OAEP  
2281 encoding of m using OAEP parameters of
- 2282 a. m = TPM\_MIGRATE\_ASYMKEY structure (step 7)
- 2283 b. pHash = TPM\_PERMANENT\_DATA -> ownerAuth
- 2284 c. seed = s1 = k1 (step 6)
22859. If generateRandom = TRUE
- 2286 a. Create r1 by obtaining values from the TPM RNG. The size of r1 MUST be the same  
2287 size as o1. Set random parameter to r1
- 228810.If generateRandom = FALSE
- 2289 a. Create r1 by applying MGF1 to the TPM Owner AuthData. The size of r1 MUST be the  
2290 same size as o1. Set randomSize to 0.
- 229111.Create x1 by XOR of o1 with r1
- 229212.Encrypt x1 with the manuMaintPub key using the TPM\_ES\_RSAESOAEP\_SHA1\_MGF1  
2293 encryption scheme.
- 229413.Set a1 -> encData to the encryption of x1
- 229514.Set TPM\_PERMANENT\_FLAGS -> maintenanceDone to TRUE
- 229615.Return a1 in the archive parameter

## 229712.2 TPM\_LoadMaintenanceArchive

### 2298Start of informative comment:

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

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

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

### 2307End of informative comment.

### 2308Incoming 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.

2309

## 2310Outgoing 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

## 2311Description

2312The maintenance mechanisms in the TPM MUST not require the TPM to hold a global  
2313secret. The definition of global secret is a secret value shared by more than one TPM.

2314The TPME is not allowed to pre-store or use unique identifiers in the TPM for the purpose of  
2315maintenance. The TPM MUST NOT use the endorsement key for identification or encryption  
2316in the maintenance process. The maintenance process MAY use a TPM Identity to deliver  
2317maintenance information to specific TPM's.

2318The maintenance process can only change the SRK, tpmProof and TPM Owner AuthData  
2319fields.

2320The maintenance process can only access data in shielded locations where this data is  
2321necessary to validate the TPM Owner, validate the TPME and manipulate the blob

2322The TPM MUST be conformant to the TPM specification, protection profiles and security  
2323targets after maintenance. The maintenance MAY NOT decrease the security values from  
2324the original security target.

2325The security target used to evaluate this TPM MUST include this command in the TOE.

## 2326Actions

2327The TPM SHALL perform the following when executing the command

23281. Validate the TPM Owner's AuthData

23292. Validate that the maintenance information was sent by the TPME. The validation  
2330 mechanism MUST use a strength of function that is at least the same strength of  
2331 function as a digital signature performed using a 2048 bit RSA key.

23323. The packet MUST contain m2 as defined in section 12.1.

23334. Ensure that only the target TPM can interpret the maintenance packet. The protection  
2334 mechanism MUST use a strength of function that is at least the same strength of  
2335 function as a digital signature performed using a 2048 bit RSA key.



23365. Execute the actions of TPM\_OwnerClear.

23376. Process the maintenance information

2338     a. Update the SRK

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

2340     b. Update TPM\_PERMANENT\_DATA -> tpmProof

2341     c. Update TPM\_PERMANENT\_DATA -> ownerAuth

23427. Set TPM\_PERMANENT\_FLAGS -> maintenanceDone to TRUE

2343

## 2344**12.3 TPM\_KillMaintenanceFeature**

### 2345**Start of informative comment:**

2346The TPM\_KillMaintenanceFeature is a permanent action that prevents ANYONE from  
2347creating a maintenance archive. This action, once taken, is permanent until a new TPM  
2348Owner is set.

2349This action is to allow those customers who do not want the maintenance feature to not  
2350allow the use of the maintenance feature.

2351At the discretion of the Owner, it should be possible to kill the maintenance feature in such  
2352a way that the only way to recover maintainability of the platform would be to wipe out the  
2353root keys. This feature is mandatory in any TPM that implements the maintenance feature.

### 2354**End of informative comment.**

### 2355**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.

### 2356**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.

### 2357**Actions**

23581. Validate the TPM Owner AuthData

23592. Set the TPM\_PERMANENT\_FLAGS.allowMaintenance flag to FALSE.

## 236012.4 TPM\_LoadManuMaintPub

### 2361Start of informative comment:

2362The TPM\_LoadManuMaintPub command loads the manufacturer's public key for use in the  
2363maintenance process. The command installs manuMaintPub in PERMANENT data storage  
2364inside a TPM. Maintenance enables duplication of non-migratory data in protected storage.  
2365There is therefore a security hole if a platform is shipped before the maintenance public key  
2366has been installed in a TPM.

2367The command is expected to be used before installation of a TPM Owner or any key in TPM  
2368protected storage. It therefore does not use authorization.

### 2369End of informative comment.

### 2370Incoming 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

### 2371Outgoing 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

### 2372Description

2373The pubKey MUST specify an algorithm whose strength is not less than the RSA algorithm  
2374with 2048bit keys.

2375pubKey SHOULD unambiguously identify the entity that will perform the maintenance  
2376process with the TPM Owner.

2377TPM\_PERMANENT\_DATA -> manuMaintPub SHALL exist in a TPM-shielded location, only.

2378If an entity (Platform Entity) does not support the maintenance process but issues a  
2379platform credential for a platform containing a TPM that supports the maintenance process,  
2380the value of TPM\_PERMANENT\_DATA -> manuMaintPub MUST be set to zero before the  
2381platform leaves the entity's control. That is, this ordinal can only be run once, and used to  
2382either load the key or load a NULL key.

### 2383**Actions**

2384The first valid TPM\_LoadManuMaintPub command received by a TPM SHALL

23851. Store the parameter pubKey as TPM\_PERMANENT\_DATA -> manuMaintPub.

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

23873. Export the checksum

23884. Subsequent calls to TPM\_LoadManuMaintPub SHALL return code  
2389 TPM\_DISABLED\_CMD.

## 2390 **12.5 TPM\_ReadManuMaintPub**

### 2391 **Start of informative comment:**

2392 The TPM\_ReadManuMaintPub command is used to check whether the manufacturer's  
2393 public maintenance key in a TPM has the expected value. This may be useful during the  
2394 manufacture process. The command returns a digest of the installed key, rather than the  
2395 key itself. This hinders discovery of the maintenance key, which may (or may not) be useful  
2396 for manufacturer privacy.

2397 The command is expected to be used before installation of a TPM Owner or any key in TPM  
2398 protected storage. It therefore does not use authorization.

### 2399 **End of Informative Comments**

### 2400 **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

### 2401 **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

### 2402 **Description**

2403 This command returns the hash of the antiReplay nonce and the previously loaded  
2404 manufacturer's maintenance public key.

### 2405 **Actions**

2406 The TPM\_ReadManuMaintPub command SHALL

2407 1. Create "checksum" by concatenating data to form (TPM\_PERMANENT\_DATA ->  
2408 manuMaintPub || antiReplay) and passing the concatenated data through SHA-1.

2409 2. Export the checksum

## 241013. Cryptographic Functions

### 241113.1 TPM\_SHA1Start

#### 2412Start of informative comment:

2413This capability starts the process of calculating a SHA-1 digest.

2414The exposure of the SHA-1 processing is a convenience to platforms in a mode that do not  
2415have sufficient memory to perform SHA-1 themselves. As such, the use of SHA-1 is  
2416restrictive on the TPM.

2417The TPM may not allow any other types of processing during the execution of a SHA-1  
2418session. There is only one SHA-1 session active on a TPM. The exclusivity of a SHA-1  
2419context is due to the relatively large volatile buffer it requires in order to hold the  
2420intermediate results between the SHA-1 context commands. This buffer can be in  
2421contradiction to other command needs.

2422After the execution of TPM\_SHA1Start, and prior to TPM\_SHA1Complete or  
2423TPM\_SHA1CompleteExtend, the receipt of any command other than TPM\_SHA1Update will  
2424cause the invalidation of the SHA-1 session.

#### 2425End of informative comment.

#### 2426Incoming 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

#### 2427Outgoing 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.

#### 2428Description

24291. This capability prepares the TPM for a subsequent TPM\_SHA1Update,  
2430 TPM\_SHA1Complete or TPM\_SHA1CompleteExtend command. The capability SHALL  
2431 open a thread that calculates a SHA-1 digest.

604

605

24322. After receipt of TPM\_SHA1Start, and prior to the receipt of TPM\_SHA1Complete or  
2433 TPM\_SHA1CompleteExtend, receipt of any command other than TPM\_SHA1Update  
2434 invalidates the SHA-1 session.
- 2435 a. If the command received is TPM\_ExecuteTransport, the SHA-1 session invalidation is  
2436 based on the wrapped command, not the TPM\_ExecuteTransport ordinal.
- 2437 b. A SHA-1 thread (start, update, complete) MUST take place either completely outside  
2438 a transport session or completely within a single transport session.

## 2439**13.2 TPM\_SHA1Update**

### 2440**Start of informative comment:**

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

### 2443**End of informative comment.**

### 2444**Incoming Operands and Sizes**

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

### 2445**Outgoing Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_SHA1Update

### 2446**Description**

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

### 2449**Actions**

24501. If there is no existing SHA-1 thread, return TPM\_SHA\_THREAD

24512. If numBytes is not a multiple of 64

2452 a. Return TPM\_SHA\_ERROR

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

24543. If numBytes is greater than maxNumBytes returned by TPM\_SHA1Start

2455 a. Return TPM\_SHA\_ERROR

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

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

2458



**245913.3 TPM\_SHA1Complete****2460Start of informative comment:**

2461This capability terminates a pending SHA-1 calculation.

**2462End of informative comment.****2463Incoming 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

**2464Outgoing 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.

**2465Description**

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

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

## 2471**13.4 TPM\_SHA1CompleteExtend**

### 2472**Start of informative comment:**

2473This capability terminates a pending SHA-1 calculation and EXTENDS the result into a  
2474Platform Configuration Register using a SHA-1 hash process.

2475This command is designed to complete a hash sequence and extend a PCR in memory-less  
2476environments.

### 2477**End of informative comment.**

### 2478**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

### 2479**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.

### 2480**Description**

2481This command SHALL incorporate a partial or complete block of data into the digest of an  
2482existing SHA-1 thread, EXTEND the resultant digest into a PCR, and terminate the SHA-1  
2483session. hashDataSize MAY have values in the range of 0 through 64, inclusive.

2484The SHA-1 session MUST terminate even if the command returns an error, e.g.  
2485TPM\_BAD\_LOCALITY.

### 2486**Actions**

24875. Validate that pcrNum represents a legal PCR number. On error, return TPM\_BADINDEX.

24886. Map V1 to TPM\_STANY\_DATA

24897. Map L1 to V1 -> localityModifier

24908. If the current locality, held in L1, is not selected in TPM\_PERMANENT\_DATA -> pcrAttrib  
2491 [pcrNum]. pcrExtendLocal, return TPM\_BAD\_LOCALITY

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

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

## 2496**13.5 TPM\_Sign**

### 2497**Start of informative comment:**

2498The Sign command signs data and returns the resulting digital signature.

2499The TPM does not allow TPM\_Sign with a TPM\_KEY\_IDENTITY (AIK) because TPM\_Sign can  
2500sign arbitrary data and could be used to fake a quote. (This could have been relaxed to  
2501allow TPM\_Sign with an AIK if the signature scheme is \_INFO For an \_INFO key, the  
2502metadata prevents TPM\_Sign from faking a quote.)

### 2503**End of informative comment.**

### 2504**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

### 2505**Outgoing Operands and Sizes**

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

**2506 Description**

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

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

**2514 Actions**

2515 1. The TPM validates the `AuthData` to use the key pointed to by `keyHandle`.

2516 2. If the `areaToSignSize` is 0 the TPM returns `TPM_BAD_PARAMETER`.

2517 3. Validate that `keyHandle` -> `keyUsage` is `TPM_KEY_SIGNING` or `TPM_KEY_LEGACY`, if not  
2518 return the error code `TPM_INVALID_KEYUSAGE`

2519 4. The TPM verifies that the signature scheme and key size can properly sign the  
2520 `areaToSign` parameter.

2521 5. If signature scheme is `TPM_SS_RSASSAPKCS1v15_SHA1` then

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

2523 b. Set `S1` to `areaToSign`

2524 6. Else if signature scheme is `TPM_SS_RSASSAPKCS1v15_DER` then

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

2527 b. Set `S1` to `areaToSign`

2528 7. else if signature scheme is `TPM_SS_RSASSAPKCS1v15_INFO` then

2529 a. Create `S2` a `TPM_SIGN_INFO` structure

2530 b. Set `S2` -> fixed to "SIGN"

2531 c. Set `S2` -> replay to `nonceOdd`

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

2534 d. Set `S2` -> `dataLen` to `areaToSignSize`

2535 e. Set `S2` -> `data` to `areaToSign`

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

2537 8. Else return `TPM_INVALID_KEYUSAGE`

2538 9. The TPM computes the signature, `sig`, using the key referenced by `keyHandle` using `S1`  
2539 as the value to sign

2540 10. Return the computed signature in `Sig`

## 2541**13.6 TPM\_GetRandom**

### 2542**Start of informative comment:**

2543TPM\_GetRandom returns the next bytesRequested bytes from the random number  
2544generator to the caller.

2545It is recommended that a TPM implement the RNG in a manner that would allow it to return  
2546RNG bytes such that the frequency of bytesRequested being more than the number of bytes  
2547available is an infrequent occurrence.

### 2548**End of informative comment.**

### 2549**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

### 2550**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

### 2551**Actions**

25521. The TPM determines if amount bytesRequested is available from the TPM.

25532. Set randomBytesSize to the number of bytes available from the RNG. This number MAY  
2554 be less than bytesRequested.

25553. Set randomBytes to the next randomBytesSize bytes from the RNG

2556**13.7 TPM\_StirRandom**2557**Start of informative comment:**

2558TPM\_StirRandom adds entropy to the RNG state.

2559**End of informative comment.**2560**Incoming Operands and Sizes**

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

2561**Outgoing Operands and Sizes**

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

2562**Actions**

25631. If dataSize is not less than 256 bytes, the TPM MAY return TPM\_BAD\_PARAMETER.

25642. The TPM updates the state of the current RNG using the appropriate mixing function.

## 2565**13.8 TPM\_CertifyKey**

### 2566**Start of informative comment:**

2567The TPM\_CertifyKey operation allows one key to certify the public portion of another key.

2568A TPM identity key may be used to certify non-migratable keys but is not permitted to  
2569certify migratory keys or certified migration keys. As such, it allows the TPM to make the  
2570statement “this key is held in a TPM-shielded location, and it will never be revealed.” For  
2571this statement to have veracity, the Challenger must trust the policies used by the entity  
2572that issued the identity and the maintenance policy of the TPM manufacturer.

2573Signing and legacy keys may be used to certify both migratable and non-migratable keys.  
2574Then the usefulness of a certificate depends on the trust in the certifying key by the  
2575recipient of the certificate.

2576The key to be certified must be loaded before TPM\_CertifyKey is called.

2577The determination to use the TPM\_CERTIFY\_INFO or TPM\_CERTIFY\_INFO2 on the output is  
2578based on which PCRs and what localities the certified key is restricted to. A key to be  
2579certified that does not have locality restrictions and which uses no PCRs greater than PCR  
2580#15 will cause this command to return and sign a TPM\_CERTIFY\_INFO structure, which  
2581provides compatibility with V1.1 TPMs.

2582When this command is run to certify all other keys (those that use PCR #16 or higher, as  
2583well as those limited by locality in any way), it will return and sign a TPM\_CERTIFY\_INFO2  
2584structure.

2585TPM\_CertifyKey does not support the case where (a) the certifying key requires a usage  
2586authorization to be provided but (b) the key-to-be-certified does not. In such cases,  
2587TPM\_CertifyKey2 must be used. TPM\_CertifyKey cannot be used to certify CMKs.

2588If a command tag (in the parameter array) specifies only one authorisation session, then the  
2589TPM convention is that the first session listed is ignored (authDataUsage must be  
2590TPM\_AUTH\_NEVER for this key) and the incoming session data is used for the second auth  
2591session in the list. In TPM\_CertifyKey, the first session is the certifying key and the second  
2592session is the key-to-be-certified. In TPM\_CertifyKey2, the first session is the key-to-be-  
2593certified and the second session is the certifying key.

2594The key handles of both the certifying key and the key to be certified are not included in the  
2595HMAC protecting the command. This permits key handle virtualization (swapping of keys  
2596in and out of the TPM that results in different key handles while at the same time  
2597maintaining key identifiers of upper layer software). In environments where the interface to  
2598the TPM is accessible by other parties, the key handles not being protected allows an  
2599attacker to change the handle of the key to be certified. This can be avoided by processing  
2600this command within a transport session and making sure that antiReplay indeed contains  
2601a nonce.

### 2602**End of informative comment.**



## 2603Incoming Operands and Sizes

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

## 2604Outgoing 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.

## 2605Actions

26061. The TPM validates that the key pointed to by certHandle has a signature scheme of  
2607 TPM\_SS\_RSASSAPKCS1v15\_SHA1 or TPM\_SS\_RSASSAPKCS1v15\_INFO

26082. Verify command and key AuthData values:

2609 a. If tag is TPM\_TAG\_RQU\_AUTH2\_COMMAND

2610 i. The TPM verifies the AuthData in certAuthHandle provides authorization to  
2611 use the key pointed to by certHandle, return TPM\_AUTHFAIL on error

2612 ii. The TPM verifies the AuthData in keyAuthHandle provides authorization to  
2613 use the key pointed to by keyHandle, return TPM\_AUTH2FAIL on error

2614 b. else if tag is TPM\_TAG\_RQU\_AUTH1\_COMMAND

2615 i. Verify that authDataUsage is TPM\_AUTH\_NEVER for the key referenced by  
2616 certHandle, return TPM\_AUTHFAIL on error.

2617 ii. The TPM verifies the AuthData in keyAuthHandle provides authorization to  
2618 use the key pointed to by keyHandle, return TPM\_AUTHFAIL on error

2619 c. else if tag is TPM\_TAG\_RQU\_COMMAND

2620 i. Verify that authDataUsage is TPM\_AUTH\_NEVER for the key referenced by  
2621 certHandle, return TPM\_AUTHFAIL on error.

658  
659

2622        ii. Verify that authDataUsage is TPM\_AUTH\_NEVER or  
2623        TPM\_NO\_READ\_PUBKEY\_AUTH for the key referenced by keyHandle, return  
2624        TPM\_AUTHFAIL on error.

26253. If keyHandle -> payload is not TPM\_PT\_ASYM, return TPM\_INVALID\_KEYUSAGE.

26264. If the key pointed to by certHandle is an identity key (certHandle -> keyUsage is  
2627    TPM\_KEY\_IDENTITY)

2628    a. If keyHandle -> keyFlags -> migratable is TRUE return TPM\_MIGRATEFAIL

26295. Validate that certHandle -> keyUsage is TPM\_KEY\_SIGN, TPM\_KEY\_IDENTITY or  
2630    TPM\_KEY\_LEGACY, if not return TPM\_INVALID\_KEYUSAGE

26316. Validate that keyHandle -> keyUsage is TPM\_KEY\_SIGN, TPM\_KEY\_STORAGE,  
2632    TPM\_KEY\_IDENTITY, TPM\_KEY\_BIND or TPM\_KEY\_LEGACY, if not return  
2633    TPM\_INVALID\_KEYUSAGE

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

2636    a. Set V1 to 1.2

26378. Else

2638    a. Set V1 to 1.1

26399. If keyHandle -> pcrInfoSize is not 0

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

2641        i. Create a digestAtRelease according to the specified TPM\_STCLEAR\_DATA ->  
2642        PCR registers and compare to keyHandle -> digestAtRelease and if a mismatch  
2643        return TPM\_WRONGPCRVAL

2644        ii. If specified validate any locality requests on error TPM\_BAD\_LOCALITY

2645    b. If V1 is 1.1

2646        i. Create C1 a TPM\_CERTIFY\_INFO structure

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

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

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

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

2652    c. Else

2653        i. Create C1 a TPM\_CERTIFY\_INFO2 structure

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

2655        iii. Set C1 -> pcrInfoSize to the size of an appropriate TPM\_PCR\_INFO\_SHORT  
2656        structure.

2657        iv. Set C1 -> pcrInfo to a properly filled out TPM\_PCR\_INFO\_SHORT structure,  
2658        using the information from keyHandle.

2659        v. Set C1 -> migrationAuthoritySize to 0

266010.Else

2661    a. Create C1 a TPM\_CERTIFY\_INFO structure

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

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

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

266612.Set C1 -> pubKeyDigest to H1

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

266814.The TPM sets certifyInfo to C1.

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

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

## 267213.9 TPM\_CertifyKey2

### 2673Start of informative comment:

2674This command is based on TPM\_CertifyKey, but includes the ability to certify a Certifiable  
2675Migration Key (CMK), which requires extra input parameters.

2676TPM\_CertifyKey2 always produces a TPM\_CERTIFY\_INFO2 structure.

2677TPM\_CertifyKey2 does not support the case where (a) the key-to-be-certified requires a  
2678usage authorization to be provided but (b) the certifying key does not.

2679If a command tag (in the parameter array) specifies only one authorisation session, then the  
2680TPM convention is that the first session listed is ignored (authDataUsage must be  
2681TPM\_NO\_READ\_PUBKEY\_AUTH or TPM\_AUTH\_NEVER for this key) and the incoming  
2682session data is used for the second auth session in the list. In TPM\_CertifyKey2, the first  
2683session is the key to be certified and the second session is the certifying key.

2684The key handles of both the certifying key and the key to be certified are not included in the  
2685HMAC protecting the command. This permits key handle virtualization (swapping of keys  
2686in and out of the TPM that results in different key handles while at the same time  
2687maintaining key identifiers of upper layer software). In environments where the interface to  
2688the TPM is accessible by other parties, the key handles not being protected allows an  
2689attacker to change the handle of the key to be certified. This can be avoided by processing  
2690this command within a transport session and making sure that antiReplay indeed contains  
2691a nonce.

### 2692End of informative comment.

### 2693Incoming 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.

## 2694 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.

## 2695 Actions

26961. The TPM validates that the key pointed to by certHandle has a signature scheme of  
2697 TPM\_SS\_RSASSAPKCS1v15\_SHA1 or TPM\_SS\_RSASSAPKCS1v15\_INFO

26982. Verify command and key AuthData values:

2699 a. If tag is TPM\_TAG\_RQU\_AUTH2\_COMMAND

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

2702 ii. The TPM verifies the AuthData in certAuthHandle provides authorization to  
2703 use the key pointed to by certHandle, return TPM\_AUTH2FAIL on error

2704 b. else if tag is TPM\_TAG\_RQU\_AUTH1\_COMMAND

2705 i. Verify that authDataUsage is TPM\_AUTH\_NEVER or  
2706 TPM\_NO\_READ\_PUBKEY\_AUTH for the key referenced by keyHandle, return  
2707 TPM\_AUTHFAIL on error

2708 ii. The TPM verifies the AuthData in certAuthHandle provides authorization to  
2709 use the key pointed to by certHandle, return TPM\_AUTH2FAIL on error

2710 c. else if tag is TPM\_TAG\_RQU\_COMMAND

2711 i. Verify that authDataUsage is TPM\_AUTH\_NEVER or  
2712 TPM\_NO\_READ\_PUBKEY\_AUTH for the key referenced by keyHandle, return  
2713 TPM\_AUTHFAIL on error

2714           ii. Verify that authDataUsage is TPM\_AUTH\_NEVER for the key referenced by  
2715           certHandle, return TPM\_AUTHFAIL on error.

27163. If the key pointed to by certHandle is an identity key (certHandle -> keyUsage is  
2717   TPM\_KEY\_IDENTITY)

2718   a. If keyHandle -> keyFlags -> migratable is TRUE and [keyHandle -> keyFlags->  
2719   migrateAuthority is FALSE or (keyHandle -> payload != TPM\_PT\_MIGRATE\_RESTRICTED  
2720   and keyHandle -> payload != TPM\_PT\_MIGRATE\_EXTERNAL)] return  
2721   TPM\_MIGRATEFAIL

27224. Validate that certHandle -> keyUsage is TPM\_KEY\_SIGNING, TPM\_KEY\_IDENTITY or  
2723   TPM\_KEY\_LEGACY, if not return TPM\_INVALID\_KEYUSAGE

27245. Validate that keyHandle -> keyUsage is TPM\_KEY\_SIGNING, TPM\_KEY\_STORAGE,  
2725   TPM\_KEY\_IDENTITY, TPM\_KEY\_BIND or TPM\_KEY\_LEGACY, if not return  
2726   TPM\_INVALID\_KEYUSAGE

27276. The TPM SHALL create a c1 a TPM\_CERTIFY\_INFO2 structure from the key pointed to  
2728   by keyHandle

27297. Create TPM\_DIGEST H1 which is the SHA-1 hash of keyHandle -> pubKey -> key. Note  
2730   that <key> is the actual public modulus, and does not include any structure formatting.

27318. Set C1 -> pubKeyDigest to H1

27329. Copy the antiReplay parameter to c1 -> data

273310. Copy other keyHandle parameters into C1

273411. If keyHandle -> payload == TPM\_PT\_MIGRATE\_RESTRICTED or  
2735   TPM\_PT\_MIGRATE\_EXTERNAL

2736   a. create thisPubKey, a TPM\_PUBKEY structure containing the public key, algorithm  
2737   and parameters corresponding to keyHandle

2738   b. Verify that the migration authorization is valid for this key

2739       i. Create M2 a TPM\_CMK\_MIGAUTH structure

2740       ii. Set M2 -> msaDigest to migrationPubDigest

2741       iii. Set M2 -> pubkeyDigest to SHA-1[thisPubKey]

2742       iv. Verify that [keyHandle -> migrationAuth] == HMAC(M2) signed by using  
2743       tpmProof as the secret and return error TPM\_MA\_SOURCE on mismatch

2744   c. Set C1 -> migrationAuthority = SHA-1(migrationPubDigest || keyHandle -> payload)

2745   d. if keyHandle -> payload == TPM\_PT\_MIGRATE\_RESTRICTED

2746       i. Set C1 -> payloadType = TPM\_PT\_MIGRATE\_RESTRICTED

2747   e. if keyHandle -> payload == TPM\_PT\_MIGRATE\_EXTERNAL

2748       i. Set C1 -> payloadType = TPM\_PT\_MIGRATE\_EXTERNAL

274912. Else

2750   a. set C1 -> migrationAuthority = NULL

2751   b. set C1 -> migrationAuthoritySize = 0



685

686

2752 c. Set C1 -> payloadType = TPM\_PT\_ASYM

2753 13. If keyHandle -> pcrInfoSize is not 0

2754 a. The TPM MUST set c1 -> pcrInfoSize to match the pcrInfoSize from the keyHandle  
2755 key.

2756 b. The TPM MUST set c1 -> pcrInfo to match the pcrInfo from the keyHandle key

2757 c. If keyHandle -> keyFlags has pcrIgnoredOnRead set to FALSE

2758 i. Create a digestAtRelease according to the specified TPM\_STCLEAR\_DATA ->  
2759 PCR registers and compare to keyHandle -> digestAtRelease and if a mismatch  
2760 return TPM\_WRONGPCRVAL

2761 ii. If specified validate any locality requests on error TPM\_BAD\_LOCALITY

2762 14. Else

2763 a. The TPM MUST set c1 -> pcrInfoSize to 0

2764 15. The TPM creates m1, a message digest formed by taking the SHA-1 of c1

2765 a. The TPM then computes a signature using certHandle -> sigScheme. The resulting  
2766 signed blob is returned in outData

## 2767**14. Endorsement Key Handling**

### 2768**Start of informative comment:**

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

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

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

### 2776**End of informative comment.**

27771. A TPM SHALL NOT install an EK unless generated on the TPM by execution of  
2778 TPM\_CreateEndorsementKeyPair or TPM\_CreateRevocableEK

## 277914.1 TPM\_CreateEndorsementKeyPair

### 2780Start of informative comment:

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

### 2783End of informative comment.

### 2784Incoming 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

### 2785Outgoing 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

### 2786Actions

27871. If an EK already exists, return TPM\_DISABLED\_CMD

27882. Validate the keyInfo parameters for the key description

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

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

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

27943. Create a key pair called the “endorsement key pair” using a TPM-protected capability.  
2795 The type and size of key are that indicated by keyInfo. Set encScheme to  
2796 TPM\_ES\_RSAESOAEP\_SHA1\_MGF1.

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

27985. Store the PRIVEK

27996. Create TPM\_PERMANENT\_DATA -> tpmDAASeed from the TPM RNG

- 28007. Create TPM\_PERMANENT\_DATA -> daaProof from the TPM RNG
- 28018. Create TPM\_PERMANENT\_DATA -> daaBlobKey from the TPM RNG
- 28029. Set TPM\_PERMANENT\_FLAGS -> CEKPUsed to TRUE
- 280310. Set TPM\_PERMANENT\_FLAGS -> enableRevokeEK to FALSE

## 2804 **14.2 TPM\_CreateRevocableEK**

### 2805 **Start of informative comment:**

2806 This command creates the TPM endorsement key. It returns a failure code if an  
2807 endorsement key already exists. The TPM vendor may have a separate mechanism to create  
2808 the EK and “squirt” the value into the TPM.

2809 The input parameters specify whether the EK is capable of being reset, whether the  
2810 AuthData value to reset the EK will be generated by the TPM, and the new AuthData value  
2811 itself if it is not to be generated by the TPM. The output parameter is the new AuthData  
2812 value that must be used when resetting the EK (if it is capable of being reset).

2813 The command TPM\_RevokeTrust must be used to reset an EK (if it is capable of being  
2814 reset).

2815 Owner authorisation is unsuitable for authorizing resetting of an EK: someone with  
2816 Physical Presence can remove a genuine Owner, install a new Owner, and revoke the EK.  
2817 The genuine Owner can reinstall, but the platform will have lost its original attestation and  
2818 may not be trusted by challengers. Therefore if a password is to be used to revoke an EK, it  
2819 must be a separate password, given to the genuine Owner.

2820 In v1.2 an OEM has extra choices when creating EKs.

2821 a) An OEM could manufacture all of its TPMs with enableRevokeEK==TRUE.

2822 If the OEM has tracked the EKreset passwords for these TPMs, the OEM can give the  
2823 passwords to customers. The customers can use the passwords as supplied, change the  
2824 passwords, or clear the EKs and create new EKs with new passwords.

2825 If EKreset passwords are random values, the OEM can discard those values and not give  
2826 them to customers. There is then a low probability (statistically zero) chance of a local DOS  
2827 attack to reset the EK by guessing the password. The chance of a remote DOS attack is zero  
2828 because Physical Presence must also be asserted to use TPM\_RevokeTrust.

2829 b) An OEM could manufacture some of its TPMs with enableRevokeEK==FALSE. Then the  
2830 EK can never be revoked, and the chance of even a local DOS attack on the EK is  
2831 eliminated.

### 2832 **End of informative comment.**

2833 This is an optional command

## 2834 Incoming Operands and Sizes

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

## 2835 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CreateRevocableEK
4	<>	3S	<>	TPM_PUBKEY	pubEndorsementKey	The public endorsement key
5	20	4S	20	TPM_DIGEST	checksum	Hash of pubEndorsementKey and antiReplay
6	20	5S	20	TPM_NONCE	outputEKreset	The AuthData value to use TPM_RevokeTrust

## 2836 Actions

28371. If an EK already exists, return TPM\_DISABLED\_CMD

28382. Perform the actions of TPM\_CreateEndorsementKeyPair, if any errors return with error

28393. Set TPM\_PERMANENT\_FLAGS -> enableRevokeEK to TRUE

2840 a. If generateReset is TRUE then

2841 i. Set TPM\_PERMANENT\_DATA -> EKreset to the next value from the TPM RNG

2842 b. Else

2843 i. Set TPM\_PERMANENT\_DATA -> EKreset to inputEKreset

28444. Return PUBEK, checksum and Ekreset

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

## 284914.3 TPM\_RevokeTrust

### 2850Start of informative comment:

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

### 2854End of informative comment.

2855This is an optional command

### 2856Incoming 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

### 2857Outgoing 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

### 2858Actions

28591. The TPM MUST validate that TPM\_PERMANENT\_FLAGS -> enableRevokeEK is TRUE,  
2860 return TPM\_PERMANENTEK on error

28612. The TPM MUST validate that the EKReset matches TPM\_PERMANENT\_DATA -> EKReset  
2862 return TPM\_AUTHFAIL on error.

28633. Ensure that physical presence is being asserted

28644. Perform the actions of TPM\_OwnerClear (excepting the command authentication)

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

2867 b. Set TPM\_PERMANENT\_FLAGS -> nvLocked to FALSE

28685. Invalidate TPM\_PERMANENT\_DATA -> tpmDAASeed

28696. Invalidate TPM\_PERMANENT\_DATA -> daaProof

28707. Invalidate TPM\_PERMANENT\_DATA -> daaBlobKey

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

## 287214.4 TPM\_ReadPubek

### 2873Start of informative comment:

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

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

### 2878End of informative comment.

### 2879Incoming 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

### 2880Outgoing 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

### 2881Description

2882This command returns the PUBEK.

### 2883Actions

2884The TPM\_ReadPubek command SHALL

28851. If TPM\_PERMANENT\_FLAGS -> readPubek is FALSE return TPM\_DISABLED\_CMD

28862. If no EK is present the TPM MUST return TPM\_NO\_ENDORSEMENT

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

28894. Export the PUBEK and checksum.



## 289014.5 TPM\_OwnerReadInternalPub

### 2891Start of informative comment:

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

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

### 2896End of informative comment.

### 2897Incoming 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.

### 2898Outgoing 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.

### 2899Actions

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

29012. If keyHandle is TPM\_KH\_EK

2902 a. Set publicPortion to PUBEK

- 29033. Else If keyHandle is TPM\_KH\_SRK
- 2904   a. Set publicPortion to the TPM\_PUBKEY of the SRK
- 29054. Else return TPM\_BAD\_PARAMETER
- 29065. Export the public key of the referenced key

2907**15. Identity Creation and Activation**2908**15.1 TPM\_Makeldentity**2909**Start of informative comment:**

2910Generate a new Attestation Identity Key (AIK).

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

2917**End of informative comment.**2918**Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Makeldentity.
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	continueSrSession	Ignored
10	20			TPM_AUTHDATA	srkAuth	The authorization session digest for the inputs and the SRK. HMAC key: srk.usageAuth.
11	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication. Session type MUST be OSAP.
		2H2	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
12	20	3H2	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
13	1	4H2	1	BOOL	continueAuthSession	Ignored
14	20		20	TPM_AUTHDATA	ownerAuth	The authorization session digest for inputs and owner. HMAC key: ownerAuth.

## 2919Outgoing Operands and Sizes

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

## 2920Description

2921The public key of the new TPM identity SHALL be identityPubKey. The private key of the  
2922new TPM identity SHALL be tpm\_signature\_key.

## 2923Properties of the new identity

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

2924

2925This capability also generates a TPM\_KEY containing the tpm\_signature\_key.

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

2928The signing of TPM\_ID\_CONTENTS is not an authorized use of the key. The  
2929TPM\_PCR\_INFO\_xxx "...AtRelease" values are not validated.

## 2930Actions

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

29331. Validate the idKeyParams parameters for the key description

739

740

- 2934 a. If the algorithm type is RSA, the key length MUST be a minimum of 2048 and MUST  
2935 use the default exponent. For interoperability the key length SHOULD be 2048.
- 2936 b. If the algorithm type is other than RSA the strength provided by the key MUST be  
2937 comparable to RSA 2048
- 2938 c. If the TPM is not designed to create a key of the requested type, return the error code  
2939 TPM\_BAD\_KEY\_PROPERTY
- 2940 d. If TPM\_PERMANENT\_FLAGS -> FIPS is TRUE then
- 2941 i. If authDataUsage specifies TPM\_AUTH\_NEVER return TPM\_NOTFIPS
- 2942 2. Use authHandle to verify that the Owner authorized all TPM\_MakeIdentity input  
2943 parameters.
- 2944 3. Use srkAuthHandle to verify that the SRK owner authorized all TPM\_MakeIdentity input  
2945 parameters.
- 2946 4. Verify that idKeyParams -> keyUsage is TPM\_KEY\_IDENTITY. If it is not, return  
2947 TPM\_INVALID\_KEYUSAGE
- 2948 5. Verify that idKeyParams -> keyFlags -> migratable is FALSE. If it is not, return  
2949 TPM\_INVALID\_KEYUSAGE
- 2950 6. Create a1 by decrypting identityAuth according to the ADIP indicated by authHandle.
- 2951 7. Set continueAuthSession and continueSRKSession to FALSE.
- 2952 8. Determine the structure version
- 2953 a. If idKeyParams -> tag is TPM\_TAG\_KEY12
- 2954 i. Set V1 to 2
- 2955 ii. Create idKey a TPM\_KEY12 structure using idKeyParams as the default values  
2956 for the structure
- 2957 b. If idKeyParams -> ver is 1.1
- 2958 i. Set V1 to 1
- 2959 ii. Create idKey a TPM\_KEY structure using idKeyParams as the default values  
2960 for the structure
- 2961 9. Set the digestAtCreation values for pcrInfo
- 2962 a. For TPM\_PCR\_INFO\_LONG include the locality of the current command
- 2963 10. Create an asymmetric key pair (identityPubKey and tpm\_signature\_key) using a TPM-  
2964 protected capability, in accordance with the algorithm specified in idKeyParams
- 2965 11. Ensure that the AuthData information in A1 is properly stored in the idKey as  
2966 usageAuth.
- 2967 12. Attach identityPubKey and tpm\_signature\_key to idKey
- 2968 13. Set idKey -> migrationAuth to TPM\_PERMANENT\_DATA -> tpmProof
- 2969 14. Ensure that all TPM\_PAYLOAD\_TYPE structures identify this key as TPM\_PT\_ASYM
- 2970 15. Encrypt the private portion of idKey using the SRK as the parent key

297116. Create a TPM\_IDENTITY\_CONTENTS structure named idContents using  
2972 labelPrivCADigest and the information from idKey  
297317. Sign idContents using tpm\_signature\_key and TPM\_SS\_RSASSAPKCS1v15\_SHA1. Store  
2974 the result in identityBinding.

## 297515.2 TPM\_ActivateIdentity

### 2976Start of informative comment:

2977The purpose of TPM\_ActivateIdentity is to twofold. The first purpose is to obtain assurance  
2978that the credential in the TPM\_SYM\_CA\_ATTESTATION is for this TPM. The second purpose  
2979is to obtain the session key used to encrypt the TPM\_IDENTITY\_CREDENTIAL.

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

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

2985Only the Owner of the TPM has the privilege of activating a TPM identity. The Owner is  
2986required to authorize the TPM\_ActivateIdentity command. The owner may authorize the  
2987command using either the TPM\_OIAP or TPM\_OSAP authorization protocols.

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

### 2990End of informative comment.

### 2991Incoming 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.

## 2992Outgoing 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.

## 2993Description

29941. The command TPM\_ActivateIdentity activates a TPM identity created using the command  
2995 TPM\_MakeIdentity.

29962. The command assumes the availability of the private key associated with the identity.  
2997 The command will verify the association between the keys during the process.

29983. The command will decrypt the input blob and extract the session key and verify the  
2999 connection between the public and private keys. The input blob can be in 1.1 or 1.2  
3000 format.

## 3001Actions

3002A Trusted Platform Module that receives a valid TPM\_ActivateIdentity command SHALL do  
3003the following:

30041. Using the authHandle field, validate the owner's AuthData to execute the command and  
3005 all of the incoming parameters.

30062. Using the idKeyAuthHandle, validate the AuthData to execute command and all of the  
3007 incoming parameters

30083. Validate that the idKey is the public key of a valid TPM identity by checking that  
3009 idKeyHandle -> keyUsage is TPM\_KEY\_IDENTITY. Return TPM\_BAD\_PARAMETER on  
3010 mismatch

30114. Create H1 the digest of a TPM\_PUBKEY derived from idKey

30125. Decrypt blob creating B1 using PRIVEK as the decryption key

30136. Determine the type and version of B1



757

758

3014 a. If B1 -> tag is TPM\_TAG\_EK\_BLOB then

3015 i. B1 is a TPM\_EK\_BLOB

3016 b. Else

3017 i. B1 is a TPM\_ASYM\_CA\_CONTENTS. As there is no tag for this structure it is  
3018 possible for the TPM to make a mistake here but other sections of the structure  
3019 undergo validation

3020 7. If B1 is a version 1.1 TPM\_ASYM\_CA\_CONTENTS then

3021 a. Compare H1 to B1 -> idDigest on mismatch return TPM\_BAD\_PARAMETER

3022 b. Set K1 to B1 -> sessionKey

3023 8. If B1 is a TPM\_EK\_BLOB then

3024 a. Validate that B1 -> ekType is TPM\_EK\_TYPE\_ACTIVATE, return TPM\_BAD\_TYPE if  
3025 not.

3026 b. Assign A1 as a TPM\_EK\_BLOB\_ACTIVATE structure from B1 -> blob

3027 c. Compare H1 to A1 -> idDigest on mismatch return TPM\_BAD\_PARAMETER

3028 d. If A1 -> pcrSelection is not NULL

3029 i. Compute a composite hash C1 using the PCR selection A1 -> pcrSelection

3030 ii. Compare C1 to A1 -> pcrInfo->digestAtRelease and return  
3031 TPM\_WRONGPCRVAL on a mismatch

3032 e. If A1 -> pcrInfo specifies a locality ensure that the appropriate locality has  
3033 been asserted, return TPM\_BAD\_LOCALITY on error

3034 f. Set K1 to A1 -> symmetricKey

3035 9. Return K1

## 3036**16. Integrity Collection and Reporting**

3037**Start of informative comment:**

3038This section deals with what commands have direct access to the PCR

3039**End of informative comment.**

30401. The TPM SHALL only allow the following commands to alter the value of  
3041 TPM\_STCLEAR\_DATA -> PCR

3042 a. TPM\_Extend

3043 b. TPM\_SHA1CompleteExtend

3044 c. TPM\_Startup

3045 d. TPM\_PCR\_Reset

3046**16.1 TPM\_Extend**3047**Start of informative comment:**

3048This adds a new measurement to a PCR

3049**End of informative comment.**3050**Incoming Operands and Sizes**

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

3051**Outgoing Operands and Sizes**

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

3052**Description**

3053Add a measurement value to a PCR

3054**Actions**

30551. Validate that pcrNum represents a legal PCR number. On error, return TPM\_BADINDEX.

30562. Map L1 to TPM\_STANY\_FLAGS -&gt; localityModifier

30573. Map P1 to TPM\_PERMANENT\_DATA -&gt; pcrAttrib [pcrNum]. pcrExtendLocal

30584. If, for the value of L1, the corresponding bit is not set in the bit map P1, return  
3059 TPM\_BAD\_LOCALITY30605. Create c1 by concatenating (TPM\_STCLEAR\_DATA -> PCR[pcrNum] || inDigest). This  
3061 takes the current PCR value and concatenates the inDigest parameter.

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

30637. Store h1 to TPM\_STCLEAR\_DATA -&gt; PCR[pcrNum]

30648. If TPM\_PERMANENT\_FLAGS -> disable is TRUE or TPM\_STCLEAR\_FLAGS -> deactivated  
3065 is TRUE

3066     a. Set outDigest to 20 bytes of 0x00

30679. Else

3068     a. Set outDigest to h1

## 306916.2 TPM\_PCRRead

### 3070Start of informative comment:

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

### 3073End of informative comment.

### 3074Incoming 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

### 3075Outgoing 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

### 3076Description

3077The TPM\_PCRRead operation returns the current contents of the named register to the  
3078caller.

### 3079Actions

30801. Validate that pcrIndex represents a legal PCR number. On error, return  
3081 TPM\_BADINDEX.

30822. Set outDigest to TPM\_STCLEAR\_DATA -> PCR[pcrIndex]

30833. Return TPM\_SUCCESS

3084

### 3085**16.3 TPM\_Quote**

#### 3086**Start of informative comment:**

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

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

3095TPM\_Quote should not use a TPM\_KEY\_SIGNING, because there is no way for the remote  
3096party to tell whether TPM\_Quote or TPM\_Sign created the signature. The exception is a  
3097TPM\_KEY\_SIGNING key with the \_INFO signature scheme, because the metadata  
3098differentiates TPM\_Sign from TPM\_Quote.

#### 3099**End of informative comment.**

#### 3100**Incoming Operands and Sizes**

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

### 3101 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Quote.
4	<>	3S	<>	TPM_PCR_COMPOSITE	pcrData	A structure containing the same indices as targetPCR, plus the corresponding current PCR values.
5	4	4S	4	UINT32	sigSize	The used size of the output area for the signature
6	<>	5S	<>	BYTE[]	sig	The signed data blob.
7	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
9	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: Key -> usageAuth.

### 3102 Actions

31031. The TPM MUST validate the AuthData to use the key pointed to by keyHandle.
31042. Validate that keyHandle -> sigScheme is TPM\_SS\_RSASSAPKCS1v15\_SHA1 or  
3105 TPM\_SS\_RSASSAPKCS1v15\_INFO, if not return TPM\_INAPPROPRIATE\_SIG.
31063. Validate that keyHandle -> keyUsage is TPM\_KEY\_SIGNING, TPM\_KEY\_IDENTITY, or  
3107 TPM\_KEY\_LEGACY, if not return TPM\_INVALID\_KEYUSAGE
31084. Validate targetPCR
- 3109 a. targetPCR is a valid TPM\_PCR\_SELECTION structure
- 3110 b. On errors return TPM\_INVALID\_PCR\_INFO
31115. Create H1 a SHA-1 hash of a TPM\_PCR\_COMPOSITE using the TPM\_STCLEAR\_DATA ->  
3112 PCR indicated by targetPCR -> pcrSelect
31136. Create Q1 a TPM\_QUOTE\_INFO structure
- 3114 a. Set Q1 -> version to 1.1.0.0
- 3115 b. Set Q1 -> fixed to "QUOT"
- 3116 c. Set Q1 -> digestValue to H1
- 3117 d. Set Q1 -> externalData to externalData
31187. Sign SHA-1 hash of Q1 using keyHandle as the signature key
31198. Return the signature in sig

## 3120**16.4 TPM\_PCR\_Reset**

### 3121**Start of informative comment:**

3122For PCR with the pcrReset attribute set to TRUE, this command resets the PCR back to the  
3123default value, this mimics the actions of TPM\_Init. The PCR may have restrictions as to  
3124which locality can perform the reset operation.

3125Sending a null pcrSelection results in an error is due to the requirement that the command  
3126actually do something. If pcrSelection is null there are no PCR to reset and the command  
3127would then do nothing.

3128For PCR that are resettable, the presence of a Trusted Operating System (TOS) can change  
3129the behavior of TPM\_PCR\_Reset. The following pseudo code shows how the behavior  
3130changes

3131At TPM\_Startup

3132 If TPM\_PCR\_ATTRIBUTES->pcrReset is FALSE

3133 Set PCR to 0x00...00

3134 Else

3135 Set PCR to 0xFF...FF

3136At TPM\_PCR\_Reset

3137 If TPM\_PCR\_ATTRIBUTES->pcrReset is TRUE

3138 If TOSPresent

3139 Set PCR to 0x00...00

3140 Else

3141 Set PCR to 0xFF...FF

3142 Else

3143 Return error

3144The above pseudocode is for example only, for the details of a specific platform, the reader  
3145must review the platform specific specification. The purpose of the above pseudocode is to  
3146show that both pcrReset and the TOSPresent bit control the value in use to when the PCR  
3147resets.

### 3148**End of informative comment.**

### 3149**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



## 3150 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

## 3151 Description

3152 This command resets PCR values back to the default value. The command MUST validate  
3153 that all PCR registers that are selected are available to be reset before resetting any PCR.  
3154 This command MUST either reset all selected PCR registers or none of the PCR registers.

## 3155 Actions

3156 1. Validate that pcrSelection is valid

3157     a. is a valid TPM\_PCR\_SELECTION structure

3158     b. pcrSelection -> pcrSelect is non-zero

3159     c. On errors return TPM\_INVALID\_PCR\_INFO

3160 2. Map L1 to TPM\_STANY\_FLAGS -> localityModifier

3161 3. For each PCR selected perform the following

3162     a. If TPM\_PERMANENT\_DATA -> pcrAttrib[pcrIndex].pcrReset is FALSE, return  
3163     TPM\_NOTRESETABLE

3164     b. If, for the value L1, the corresponding bit is clear in the bit map  
3165     TPM\_PERMANENT\_DATA -> pcrAttrib[pcrIndex].pcrResetLocal, return TPM\_NOTLOCAL

3166 4. For each PCR selected perform the following

3167     a. The PCR MAY only reset to 0x00...00 or 0xFF...FF

3168     b. The logic to determine which value to use MUST be described by a platform specific  
3169     specification

## 317016.5 TPM\_Quote2

### 3171Start of informative comment:

3172The TPM\_Quote2 operation provides cryptographic reporting of PCR values. A loaded key is  
3173required for operation. TPM\_Quote2 uses a key to sign a statement that names the current  
3174value of a chosen PCR and externally supplied data (which may be a nonce supplied by a  
3175Challenger).

3176The term "externalData" is used because an important use of TPM\_Quote2 is to provide a  
3177digital signature on arbitrary data, where the signature includes the PCR values of the  
3178platform at time of signing. Hence the "externalData" is not just for anti-replay purposes,  
3179although it is (of course) used for that purpose in an integrity challenge.

3180TPM\_Quote2 differs from TPM\_Quote in that TPM\_Quote2 uses TPM\_PCR\_INFO\_SHORT to  
3181hold information relative to the PCR registers. TPM\_PCR\_INFO\_SHORT includes locality  
3182information to provide the requestor a more complete view of the current platform  
3183configuration.

### 3184End of informative comment.

### 3185Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Quote2
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can sign the PCR values.
5	20	2S	20	TPM_NONCE	externalData	160 bits of externally supplied data (typically a nonce provided by a server to prevent replay-attacks)
6	<>	3S	<>	TPM_PCR_SELECTION	targetPCR	The indices of the PCRs that are to be reported.
7	1	4S	1	BOOL	addVersion	When TRUE add TPM_CAP_VERSION_INFO to the output
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
9	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
11	20			TPM_AUTHDATA	privAuth	The authorization session digest for inputs and keyHandle. HMAC key: key -> usageAuth.

### 3186 Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Quote2
4	<>	3S	<>	TPM_PCR_INFO_SHORT	pcrData	The value created and signed for the quote
5	4	4S	4	UINT32	versionInfoSize	Size of the version info
6	<>	5S	<>	TPM_CAP_VERSION_INFO	versionInfo	The version info
7	4	6S	4	UINT32	sigSize	The used size of the output area for the signature
8	<>	7S	<>	BYTE[]	sig	The signed data blob.
9	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
10	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
11	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: Key -> usageAuth.

### 3187 Actions

31881. The TPM MUST validate the AuthData to use the key pointed to by keyHandle.
31892. Validate that keyHandle -> sigScheme is TPM\_SS\_RSASSAPKCS1v15\_SHA1 or  
3190 TPM\_SS\_RSASSAPKCS1v15\_INFO, if not return TPM\_INAPPROPRIATE\_SIG.
31913. Validate that keyHandle -> keyUsage is TPM\_KEY\_SIGNING, TPM\_KEY\_IDENTITY, or  
3192 TPM\_KEY\_LEGACY, if not return TPM\_INVALID\_KEYUSAGE
31934. Validate targetPCR is a valid TPM\_PCR\_SELECTION structure, on errors return  
3194 TPM\_INVALID\_PCR\_INFO
31955. Create H1 a SHA-1 hash of a TPM\_PCR\_COMPOSITE using the TPM\_STCLEAR\_DATA ->  
3196 PCR[] indicated by targetPCR -> pcrSelect
31976. Create S1 a TPM\_PCR\_INFO\_SHORT
- 3198 a. Set S1->pcrSelection to targetPCR
- 3199 b. Set S1->localityAtRelease to TPM\_STANY\_DATA -> localityModifier
- 3200 c. Set S1->digestAtRelease to H1
32017. Create Q1 a TPM\_QUOTE\_INFO2 structure
- 3202 a. Set Q1 -> fixed to "QUT2"
- 3203 b. Set Q1 -> infoShort to S1
- 3204 c. Set Q1 -> externalData to externalData
32058. If addVersion is TRUE
- 3206 a. Concatenate to Q1 a TPM\_CAP\_VERSION\_INFO structure

- 3207    b. Set the output parameters for versionInfo
- 32089. Else
- 3209    a. Set versionInfoSize to 0
- 3210    b. Return no bytes in versionInfo
- 321110. Sign a SHA-1 hash of Q1 using keyHandle as the signature key
- 321211. Return the signature in sig

## 3213 **17. Changing AuthData**

### 3214 **17.1 TPM\_ChangeAuth**

#### 3215 **Start of informative comment:**

3216 The TPM\_ChangeAuth command allows the owner of an entity to change the AuthData for  
3217 the entity.

3218 This command cannot invalidate the old entity. Therefore, the authorization change is only  
3219 effective if the application can guarantee that the old entity can be securely destroyed. If  
3220 not, two valid entities will exist, one with the old and one with the new authorization secret.

3221 If this command is delegated, the delegated party can expand its key use privileges. That  
3222 party can create a copy of the key with known authorization, and it can then use the key  
3223 without any ordinal restrictions.

3224 TPM\_ChangeAuth requires the encryption of one parameter (“NewAuth”). For the sake of  
3225 uniformity with other commands that require the encryption of more than one parameter,  
3226 the parameters used for used encryption are generated from the authLastNonceEven  
3227 (created during the OSAP session), nonceOdd, and the session shared secret.

3228 The parameter list to this command must always include two authorization sessions,  
3229 regardless of the state of authDataUsage for the respective keys.

#### 3230 **End of informative comment.**

3231

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

## 3232 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

## 3233 Description

32341. The parentAuthHandle session type MUST be TPM\_PID\_OSAP.

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

## 3237 Actions

32381. Verify that entityType is one of TPM\_ET\_DATA, TPM\_ET\_KEY and return the error  
3239 TPM\_WRONG\_ENTITYTYPE if not.

32402. Verify that parentAuthHandle session type is TPM\_PID\_OSAP return TPM\_BAD\_MODE  
3241 on error

32423. Verify that entityAuthHandle session type is TPM\_PID\_OIAP return TPM\_BAD\_MODE on  
3243 error

32444. If protocolID is not TPM\_PID\_ADCP, the TPM MUST return TPM\_BAD\_PARAMETER.

32455. The encData parameter MUST be the encData field from either the TPM\_STORED\_DATA  
3246 or TPM\_KEY structures.

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

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

32508. Validate that parentHandle -> keyUsage is TPM\_KEY\_STORAGE, if not return  
3251 TPM\_INVALID\_KEYUSAGE

32529. After parameter validation, the TPM creates b1 by decrypting encData using the key  
3253 pointed to by parentHandle.
325410. The TPM MUST validate that b1 is a valid TPM structure, either a  
3255 TPM\_STORE\_ASYMKEY or a TPM\_SEALED\_DATA
- 3256 a. Check the length and payload, return TPM\_INVALID\_STRUCTURE on any mismatch
- 3257 b. The TPM must validate the command using the authorization data entityAuth  
3258 parameter. The HMAC key is TPM\_STORE\_ASYMKEY -> usageAuth or  
3259 TPM\_SEALED\_DATA -> authData.
326011. The TPM replaces the AuthData for b1 with decryptAuth created above.
326112. The TPM encrypts b1 using the appropriate mechanism for the type using the  
3262 parentKeyHandle to provide the key information.
326313. The TPM MUST enforce the destruction of both the parentAuthHandle and  
3264 entityAuthHandle sessions.



## 326517.2 TPM\_ChangeAuthOwner

### 3266Start of informative comment:

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

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

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

### 3272End of informative comment.

### 3273Incoming 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.

### 3274Outgoing 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

## 3275Actions

32761. The TPM MUST validate the command using the AuthData in the ownerAuth parameter
32772. The ownerAuthHandle session type MUST be TPM\_PID\_OSAF
32783. If protocolID is not TPM\_PID\_ADFP, the TPM MUST return TPM\_BAD\_PARAMETER.
32794. Verify that entityType is either TPM\_ET\_OWNER or TPM\_ET\_SRK, and return the error  
3280 TPM\_WRONG\_ENTITYTYPE if not.
32815. Create decryptAuth by decrypting newAuth according to the ADIP indicated by  
3282 ownerAuthHandle.
32836. The TPM MUST enforce the destruction of the ownerAuthHandle session upon  
3284 completion of this command (successful or unsuccessful). This includes setting  
3285 continueAuthSession to FALSE
32867. Set the AuthData for the indicated entity to decryptAuth
32878. The TPM MUST invalidate all owner authorized OSAF and DSAP sessions, active or  
3288 saved.
32899. The TPM MAY invalidate all sessions, active or saved

## 3290 **18. Authorization Sessions**

### 3291 **18.1 TPM\_OIAP**

#### 3292 **Incoming Operands and Sizes**

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

#### 3293 **Outgoing Operands and Sizes**

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

#### 3294 **Actions**

3295 1. The TPM\_OIAP command allows the creation of an authorization session handle and the  
3296 tracking of the handle by the TPM. The TPM generates the handle and nonce.

3297 2. The TPM has an internal limit as to the number of handles that may be open at one  
3298 time, so the request for a new handle may fail if there is insufficient space available.

3299 3. Internally the TPM will do the following:

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

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

3303 4. On each subsequent use of the OIAP session the TPM MUST generate a new nonceEven  
3304 value.

3305 5. When TPM\_OIAP is wrapped in an encrypted transport session, no input or output  
3306 parameters are encrypted.

3307

## 330818.1.1 Actions to validate an OIAP session

### 3309Start of informative comment:

3310This section describes the authorization-related actions of a TPM when it receives a  
3311command that has been authorized with the OIAP protocol.

3312Many commands use OIAP authorization. The following description is therefore necessarily  
3313abstract.

### 3314End of informative comment.

## 3315Actions

3316The TPM MUST perform the following operations:

33171. The TPM MUST verify that the authorization session handle (H, say) referenced in the  
3318 command points to a valid session. If it does not, the TPM returns the error code  
3319 TPM\_INVALID\_AUTHHANDLE

33202. The TPM SHALL retrieve the latest version of the caller's nonce (nonceOdd) and  
3321 continueAuthSession flag from the input parameter list, and store it in internal TPM  
3322 memory with the authSession 'H'.

33233. The TPM SHALL retrieve the latest version of the TPM's nonce stored with the  
3324 authorization session H (authLastNonceEven) computed during the previously executed  
3325 command.

33264. The TPM MUST retrieve the secret AuthData (SecretE, say) of the target entity. The entity  
3327 and its secret must have been previously loaded into the TPM.

3328 a. If the command using the OIAP session requires owner authorization

3329 i. If TPM\_STCLEAR\_DATA -> ownerReference is TPM\_KH\_OWNER, the secret  
3330 AuthData is TPM\_PERMANENT\_DATA -> ownerAuth

3331 ii. If TPM\_STCLEAR\_DATA -> ownerReference is pointing to a delegate row

3332 (1) Set R1 a row index to TPM\_STCLEAR\_DATA -> ownerReference

3333 (2) Set D1 a TPM\_DELEGATE\_TABLE\_ROW to TPM\_PERMANENT\_DATA ->  
3334 delegateTable -> delRow[R1]

3335 (3) Set the secret AuthData to D1 -> authValue

3336 (4) Validate the TPM\_DELEGATE\_PUBLIC D1 -> pub

3337 (a) Validate D1 -> pub -> permissions based on the command ordinal

3338 (b) Validate D1 -> pub -> pcrInfo based on the PCR values

33395. The TPM SHALL perform a HMAC calculation using the entity secret data, ordinal, input  
3340 command parameters and authorization parameters per Part 1 Object-Independent  
3341 Authorization Protocol.

33426. The TPM SHALL compare HM to the AuthData value received in the input parameters. If  
3343 they are different, the TPM returns the error code TPM\_AUTHFAIL if the authorization  
3344 session is the first session of a command, or TPM\_AUTH2FAIL if the authorization

847

848

3345 session is the second session of a command. Otherwise, the TPM executes the command  
3346 which (for this example) produces an output that requires authentication.

33477. The TPM SHALL generate a nonce (nonceEven).

33488. The TPM creates an HMAC digest to authenticate the return code, return values and  
3349 authorization parameters to the same entity secret per Part 1 Object-Independent  
3350 Authorization Protocol.

33519. The TPM returns the return code, output parameters, authorization parameters and  
3352 authorization session digest.

335310. If the output continueUse flag is FALSE, then the TPM SHALL terminate the session.  
3354 Future references to H will return an error.

335511. Each time that access to an entity (e.g., key) is authorized using OIAP, the TPM MUST  
3356 validate the TPM\_PCR\_INFO\_xxx "...AtRelease" values if specified for the entity

3357 a. The TPM SHOULD validate the values before using the shared secret to validate the  
3358 command parameters. This prevents a dictionary attack on the shared secret when the  
3359 values are invalid for the entity.

## 336018.2 TPM\_OSAP

### 3361Start of informative comment:

3362The TPM\_OSAP command creates the authorization session handle, the shared secret and  
3363generates nonceEven and nonceEvenOSAP.

### 3364End of informative comment.

### 3365Incoming 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.

### 3366Outgoing 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.

### 3367Description

33681. The TPM\_OSAP command allows the creation of an authorization session handle and the  
3369 tracking of the handle by the TPM. The TPM generates the handle, nonceEven and  
3370 nonceEvenOSAP.

33712. The TPM has an internal limit on the number of handles that may be open at one time,  
3372 so the request for a new handle may fail if there is insufficient space available.

33733. The TPM\_OSAP allows the binding of an authorization to a specific entity. This allows  
3374 the caller to continue to send in AuthData for each command but not have to request  
3375 the information or cache the actual AuthData.

33764. When TPM\_OSAP is wrapped in an encrypted transport session, no input or output  
3377 parameters are encrypted.

33785. If the owner pointer is pointing to a delegate row, the TPM internally MUST treat the  
3379 OSAP session as a DSAP session

33806. TPM\_ET\_SRK or TPM\_ET\_KEYHANDLE with a value of TPM\_KH\_SRK MUST specify the  
3381 SRK.

33827. If the entity is tied to PCR values, the PCR's are not validated during the TPM\_OSAP  
3383 ordinal session creation. The PCR's are validated when the OSAP session is used.

#### 3384 Actions

33851. The TPM creates S1 a storage area that keeps track of the information associated with  
3386 the authorization.

33872. S1 MUST track the following information

3388 a. Protocol identification (i.e. TPM\_PID\_OSAP)

3389 b. nonceEven

3390 i. Initialized to the next value from the TPM RNG

3391 c. shared secret

3392 d. ADIP encryption scheme from TPM\_ENTITY\_TYPE entityType

3393 e. Any other internal TPM state the TPM needs to manage the session

33943. The TPM MUST create and MAY track the following information

3395 a. nonceEvenOSAP

3396 i. Initialized to the next value from the TPM RNG

33974. The TPM calculates the shared secret using an HMAC calculation. The key for the HMAC  
3398 calculation is the secret AuthData assigned to the key handle identified by entityType.  
3399 The input to the HMAC calculation is the concatenation of nonces nonceEvenOSAP and  
3400 nonceOddOSAP. The output of the HMAC calculation is the shared secret which is saved  
3401 in the authorization area associated with authHandle

34025. Check if the ADIP encryption scheme specified by entityType is supported, if not return  
3403 TPM\_INAPPROPRIATE\_ENC.

34046. If entityType = TPM\_ET\_KEYHANDLE

3405 a. The entity to authorize is a key held in the TPM. entityType contains the keyHandle  
3406 that holds the key.

3407 b. If entityType is TPM\_KH\_OPERATOR return TPM\_BAD\_HANDLE

34087. else if entityType = TPM\_ET\_OWNER

3409 a. This value indicates that the entity is the TPM owner. entityType is ignored

3410 b. The HMAC key is the secret pointed to by ownerReference (owner secret or delegated  
3411 secret)

34128. else if entityType = TPM\_ET\_SRK

3413 a. The entity to authorize is the SRK. entityType is ignored.

34149. else if entityType = TPM\_ET\_COUNTER

3415 a. The entity is a monotonic counter, entityType contains the counter handle

341610. else if entityType = TPM\_ET\_NV

- 3417     a. The entity is a NV index, entityValue contains the NV index  
341811.else return TPM\_BAD\_PARAMETER
- 341912.On each subsequent use of the OSAP session the TPM MUST generate a new nonce  
3420     value.
- 342113.The TPM MUST ensure that OSAP shared secret is only available while the OSAP session  
3422     is valid.
- 342314.The session MUST terminate upon any of the following conditions:
- 3424     a. The command that uses the session returns an error  
3425     b. The resource is evicted from the TPM or otherwise invalidated  
3426     c. The session is used in any command for which the shared secret is used to encrypt  
3427     an input parameter (TPM\_ENCAUTH)  
3428     d. The TPM Owner is cleared  
3429     e. TPM\_ChangeAuthOwner is executed and this session is attached to the owner  
3430     authorization  
3431     f. The session explicitly terminated with continueAuth, TPM\_Reset or  
3432     TPM\_FlushSpecific  
3433     g. All OSAP sessions associated with the delegation table MUST be invalidated when  
3434     any of the following commands execute:  
3435         i. TPM\_Delegate\_Manage  
3436         ii. TPM\_Delegate\_CreateOwnerDelegation with Increment==TRUE  
3437         iii. TPM\_Delegate\_LoadOwnerDelegation



## 3439 18.2.1 Actions to validate an OSAP session

### 3440 **Start of informative comment:**

3441 This section describes the authorization-related actions of a TPM when it receives a  
3442 command that has been authorized with the OSAP protocol.

3443 Many commands use OSAP authorization. The following description is therefore necessarily  
3444 abstract.

### 3445 **End of informative comment**

### 3446 **Actions**

3447 1. On reception of a command with ordinal C1 that uses an authorization session, the TPM  
3448 SHALL perform the following actions:

3449 2. The TPM MUST have been able to retrieve the shared secret (Shared, say) of the target  
3450 entity when the authorization session was established with TPM\_OSAP. The entity and  
3451 its secret must have been previously loaded into the TPM.

3452 3. The TPM MUST verify that the authorization session handle (H, say) referenced in the  
3453 command points to a valid session. If it does not, the TPM returns the error code  
3454 TPM\_INVALID\_AUTHHANDLE.

3455 4. The TPM MUST calculate the HMAC (HM1, say) of the command parameters according  
3456 to Part 1 Object-Specific Authorization Protocol.

3457 5. The TPM SHALL compare HM1 to the AuthData value received in the command. If they  
3458 are different, the TPM returns the error code TPM\_AUTHFAIL if the authorization session  
3459 is the first session of a command, or TPM\_AUTH2FAIL if the authorization session is the  
3460 second session of a command., the TPM executes command C1 which produces an  
3461 output (O, say) that requires authentication and uses a particular return code (RC, say).

3462 6. The TPM SHALL generate the latest version of the even nonce (nonceEven).

3463 7. The TPM MUST calculate the HMAC (HM2) of the return parameters according to section  
3464 Part 1 Object-Specific Authorization Protocol.

3465 8. The TPM returns HM2 in the parameter list.

3466 9. The TPM SHALL retrieve the continue flag from the received command. If the flag is  
3467 FALSE, the TPM SHALL terminate the session and destroy the thread associated with  
3468 handle H.

3469 10. If the shared secret was used to provide confidentiality for data in the received  
3470 command, the TPM SHALL terminate the session and destroy the thread associated with  
3471 handle H.

3472 11. Each time that access to an entity (e.g., key) is authorized using OSAP, the TPM MUST

3473 a. ensure that the OSAP shared secret is that derived from the entity using TPM\_OSAP

3474 b. validate the TPM\_PCR\_INFO\_XXX "...AtRelease" values if specified for the entity

3475 i. The TPM SHOULD validate the values before using the shared secret to  
3476 validate the command parameters. This prevents a dictionary attack on the  
3477 shared secret when the values are invalid for the entity.

3478**18.3 TPM\_DSAP**3479**Start of informative comment:**

3480The TPM\_DSAP command creates the authorization session handle using a delegated  
3481AuthData value passed into the command as an encrypted blob or from the internal  
3482delegation table. It can be used to start an authorization session for a user key or the  
3483owner.

3484As in TPM\_OSAP, it generates a shared secret and generates nonceEven and  
3485nonceEvenOSAP.

3486**End of informative comment.**3487**Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_DSAP.
4	2			TPM_ENTITY_TYPE	entityType	The type of delegation information to use
5	4			TPM_KEY_HANDLE	keyHandle	Key for which delegated authority corresponds, or 0 if delegated owner activity. Only relevant if 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

3488**Outgoing Operands and Sizes**

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

3489**Description**

34901. The TPM\_DSAP command allows the creation of an authorization session handle and the  
3491 tracking of the handle by the TPM. The TPM generates the handle, nonceEven and  
3492 nonceEvenOSAP.

34932. The TPM has an internal limit on the number of handles that may be open at one time,  
3494 so the request for a new handle may fail if there is insufficient space available.
34953. The TPM\_DSAP allows the binding of a delegated authorization to a specific entity. This  
3496 allows the caller to continue to send in AuthData for each command but not have to  
3497 request the information or cache the actual AuthData.
34984. Each ordinal that uses the DSAP session MUST validate that TPM\_PERMANENT\_DATA  
3499 -> restrictDelegate does not restrict delegation, based on keyHandle -> keyUsage and  
3500 keyHandle -> keyFlags, return TPM\_INVALID\_KEYUSAGE on error.
35015. On each subsequent use of the DSAP session the TPM MUST generate a new nonce  
3502 value and check if the ordinal to be executed has delegation to execute. The TPM MUST  
3503 ensure that the DSAP shared secret is only available while the DSAP session is valid.
35046. When TPM\_DSAP is wrapped in an encrypted transport session
- 3505 a. For input the only parameter encrypted is entityValue
- 3506 b. For output no parameters are encrypted
35077. The DSAP session MUST terminate under any of the following conditions
- 3508 a. The command that uses the session returns an error
- 3509 b. If attached to a key, when the key is evicted from the TPM or otherwise invalidated
- 3510 c. The session is used in any command for which the shared secret is used to encrypt  
3511 an input parameter (TPM\_ENCAUTH)
- 3512 d. The TPM Owner is cleared
- 3513 e. TPM\_ChangeAuthOwner is executed and this session is attached to the owner  
3514 authorization
- 3515 f. The session explicitly terminated with continueAuth, TPM\_Reset or  
3516 TPM\_FlushSpecific
- 3517 g. All DSAP sessions MUST be invalidated when any of the following commands  
3518 execute:
- 3519 i. TPM\_Delegate\_CreateOwnerDelegation
- 3520 ii. When Increment is TRUE
- 3521 iii. TPM\_Delegate\_LoadOwnerDelegation
- 3522 iv. TPM\_Delegate\_Manage
- 3523entityType = TPM\_ET\_DEL\_OWNER\_BLOB
- 3524 The entityValue parameter contains an owner delegation blob structure.
- 3525entityType = TPM\_ET\_DEL\_ROW
- 3526 The entityValue parameter contains a row number in the nv Delegation table which  
3527 should be used for the AuthData value.
- 3528entityType = TPM\_DEL\_KEY\_BLOB
- 3529 The entityValue parameter contains a key delegation blob structure.

**3530 Actions**

3531 1. If entityType == TPM\_ET\_DEL\_OWNER\_BLOB

3532     a. Map entityValue to B1 a TPM\_DELEGATE\_OWNER\_BLOB

3533     b. Validate that B1 is a valid TPM\_DELEGATE\_OWNER\_BLOB, return  
3534     TPM\_WRONG\_ENTITYTYPE on error

3535     c. Locate B1 -> pub -> familyID in the TPM\_FAMILY\_TABLE and set familyRow to  
3536     indicate row, return TPM\_BADINDEX if not found

3537     d. Set FR to TPM\_FAMILY\_TABLE.famTableRow[familyRow]

3538     e. If FR -> flags TPM\_FAMFLAG\_ENABLED is FALSE, return TPM\_DISABLED\_CMD

3539     f. Verify that B1->verificationCount equals FR -> verificationCount.

3540     g. Validate the integrity of the blob

3541         i. Copy B1 -> integrityDigest to H2

3542         ii. Set B1 -> integrityDigest to all zeros

3543         iii. Create H3 the HMAC of B1 using tpmProof as the secret

3544         iv. Compare H2 to H3 return TPM\_AUTHFAIL on mismatch

3545     h. Create S1 a TPM\_DELEGATE\_SENSITIVE by decrypting B1 -> sensitiveArea using  
3546     TPM\_DELEGATE\_KEY

3547     i. Validate S1 values

3548         i. S1 -> tag is TPM\_TAG\_DELEGATE\_SENSITIVE

3549         ii. Return TPM\_BAD\_DELEGATE on error

3550     j. Set A1 to S1 -> authValue

3551 2. Else if entityType == TPM\_ET\_DEL\_ROW

3552     a. Verify that entityValue points to a valid row in the delegation table.

3553     b. Set D1 to the delegation information in the row.

3554     c. Set A1 to D1->authValue.

3555     d. Locate D1 -> familyID in the TPM\_FAMILY\_TABLE and set familyRow to indicate that  
3556     row, return TPM\_BADINDEX if not found

3557     e. Set FR to TPM\_FAMILY\_TABLE.famTableRow[familyRow]

3558     f. If FR -> flags TPM\_FAMFLAG\_ENABLED is FALSE, return TPM\_DISABLED\_CMD

3559     g. Verify that D1->verificationCount equals FR -> verificationCount.

3560 3. Else if entityType == TPM\_ET\_DEL\_KEY\_BLOB

3561     a. Map entityValue to K1 a TPM\_DELEGATE\_KEY\_BLOB

3562     b. Validate that K1 is a valid TPM\_DELEGATE\_KEY\_BLOB, return  
3563     TPM\_WRONG\_ENTITYTYPE on error

3564     c. Locate K1 -> pub -> familyID in the TPM\_FAMILY\_TABLE and set familyRow to  
3565     indicate that row, return TPM\_BADINDEX if not found

- 3566 d. Set FR to TPM\_FAMILY\_TABLE.famTableRow[familyRow]
- 3567 e. If FR -> flags TPM\_FAMFLAG\_ENABLED is FALSE, return TPM\_DISABLED\_CMD
- 3568 f. Verify that K1 -> pub -> verificationCount equals FR -> verificationCount.
- 3569 g. Validate the integrity of the blob
  - 3570 i. Copy K1 -> integrityDigest to H2
  - 3571 ii. Set K1 -> integrityDigest to all zeros
  - 3572 iii. Create H3 the HMAC of K1 using tpmProof as the secret
  - 3573 iv. Compare H2 to H3 return TPM\_AUTHFAIL on mismatch
- 3574 h. Validate that K1 -> pubKeyDigest identifies keyHandle, return TPM\_KEYNOTFOUND
- 3575 on error
- 3576 i. Create S1 a TPM\_DELEGATE\_SENSITIVE by decrypting K1 -> sensitiveArea using
- 3577 TPM\_DELEGATE\_KEY
- 3578 j. Validate S1 values
  - 3579 i. S1 -> tag is TPM\_TAG\_DELEGATE\_SENSITIVE
  - 3580 ii. Return TPM\_BAD\_DELEGATE on error
- 3581 k. Set A1 to S1 -> authValue
- 35824. Else return TPM\_BAD\_PARAMETER
- 35835. Generate a new authorization session handle and reserve space to save protocol
- 3584 identification, shared secret, pcrInfo, both nonces, ADIP encryption scheme, delegated
- 3585 permission bits and any other information the TPM needs to manage the session.
- 35866. Read two new values from the RNG to generate nonceEven and nonceEvenOSAP.
- 35877. The TPM calculates the shared secret using an HMAC calculation. The key for the HMAC
- 3588 calculation is A1. The input to the HMAC calculation is the concatenation of nonces
- 3589 nonceEvenOSAP and nonceOddOSAP. The output of the HMAC calculation is the shared
- 3590 secret which is saved in the authorization area associated with authHandle.

## 3591 18.4 TPM\_SetOwnerPointer

### 3592 Start of informative comment:

3593 This command will set a reference to which secret the TPM will use when executing an  
3594 owner secret related OIAP or OSAP session.

3595 This command should only be used to provide an owner delegation function for legacy code  
3596 that does not itself support delegation. Normally, TPM\_STCLEAR\_DATA->ownerReference  
3597 points to TPM\_KH\_OWNER, indicating that OIAP and OSAP sessions should use the owner  
3598 authorization. This command allows ownerReference to point to an index in the delegation  
3599 table, indicating that OIAP and OSAP sessions should use the delegation authorization.

3600 In use, a TSS supporting delegation would create and load the owner delegation and set the  
3601 owner pointer to that delegation. From then on, a legacy TSS application would use its OIAP  
3602 and OSAP sessions with the delegated owner authorization.

3603 Since this command is not authorized, the ownerReference is open to DoS attacks.  
3604 Applications can attempt to recover from a failing owner authorization by resetting  
3605 ownerReference to an appropriate value.

3606 This command intentionally does not clear OSAP sessions. A TPM 1.1 application gets the  
3607 benefit of owner delegation, while the original owner can use a pre-existing OSAP session  
3608 with the actual owner authorization.

### 3609 End of informative comment.

### 3610 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

### 3611 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

### 3612 Actions

36131. Map TPM\_STCLEAR\_DATA to V1

36142. If entityType = TPM\_ET\_DEL\_ROW

- 3615 a. This value indicates that the entity is a delegate row. entityValue is a delegate index  
3616 in the delegation table.
- 3617 b. Validate that entityValue points to a legal row within the delegate table stored within  
3618 the TPM. If not return TPM\_BADINDEX
- 3619 i. Set D1 to the delegation information in the row.
- 3620 c. Locate D1 -> familyID in the TPM\_FAMILY\_TABLE and set familyRow to indicate that  
3621 row, return TPM\_BADINDEX if not found.
- 3622 d. Set FR to TPM\_FAMILY\_TABLE.famTableRow[familyRow]
- 3623 e. If FR -> flags TPM\_FAMFLAG\_ENABLED is FALSE, return TPM\_DISABLED\_CMD
- 3624 f. Verify that B1->verificationCount equals FR -> verificationCount.
- 3625 g. The TPM sets V1-> ownerReference to entityValue
- 3626 h. Return TPM\_SUCCESS
36273. else if entityType = TPM\_ET\_OWNER
- 3628 a. This value indicates that the entity is the TPM owner. entityValue is ignored.
- 3629 b. The TPM sets V1-> ownerReference to TPM\_KH\_OWNER
- 3630 c. Return TPM\_SUCCESS
36314. Return TPM\_BAD\_PARAMETER



## 3632 **19. Delegation Commands**

### 3633 **19.1 TPM\_Delegate\_Manage**

#### 3634 **Start of informative comment:**

3635 TPM\_Delegate\_Manage is the fundamental process for managing the Family tables,  
3636 including enabling/disabling Delegation for a selected Family. Normally  
3637 TPM\_Delegate\_Manage must be executed at least once (to create Family tables for a  
3638 particular family) before any other type of Delegation command in that family can succeed.

3639 TPM\_Delegate\_Manage is authorized by the TPM Owner if an Owner is installed, because  
3640 changing a table is a privileged Owner operation. If no Owner is installed,  
3641 TPM\_Delegate\_Manage requires no privilege to execute. This does not disenfranchise an  
3642 Owner, since there is no Owner, and simplifies loading of tables during platform  
3643 manufacture or on first-boot. Burn-out of TPM non-volatile storage by inappropriate use is  
3644 mitigated by the TPM's normal limits on NV-writes in the absence of an Owner. Tables can  
3645 be locked after loading, to prevent subsequent tampering, and only unlocked by the Owner,  
3646 his delegate, or the act of removing the Owner (even if there is no Owner).

3647 TPM\_Delegate\_Manage command is customized by opCode:

3648 (1) TPM\_FAMILY\_ENABLE enables/disables use of a family and all the rows of the delegate  
3649 table belonging to that family,

3650 (2) TPM\_FAMILY\_ADMIN can be used to prevent further management of the Tables until an  
3651 Owner is installed, or until the Owner is removed from the TPM. (Note that the Physical  
3652 Presence command TPM\_ForceClear always enables further management, even if  
3653 TPM\_ForceClear is used when no Owner is installed.)

3654 (3) TPM\_FAMILY\_CREATE creates a new family. Sessions are invalidated even in this case  
3655 because the lastFamilyID could wrap.

3656 (4) TPM\_FAMILY\_INVALIDATE invalidates an existing family. The TPM\_SELFTEST\_FAILED  
3657 error code is returned because the familyRow has already been validated earlier. Failure  
3658 here indicates a malfunction of the TPM.

3659 The rationale for Action 19.1 is that invalidating the family ID prevents the use of  
3660 associated delegate rows, whether or not those delegate rows are themselves invalidated.  
3661 Omitting the delegate row invalidation avoids an NV write.

#### 3662 **End of informative comment.**

### 3663Incoming 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.

### 3664Outgoing 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.

### 3665Action

36661. If opCode != TPM\_FAMILY\_CREATE

3667 a. Locate familyID in the TPM\_FAMILY\_TABLE and set familyRow to indicate row,  
3668 return TPM\_BADINDEX if not found

3669 b. Set FR, a TPM\_FAMILY\_TABLE\_ENTRY, to TPM\_FAMILY\_TABLE. famTableRow  
3670 [familyRow]

36712. If tag = TPM\_TAG\_RQU\_AUTH1\_COMMAND

3672 a. Validate the command and parameters using ownerAuth, return TPM\_AUTHFAIL on  
3673 error

910  
911

3674 b. If the command is delegated (authHandle session type is TPM\_PID\_DSAP or through  
3675 ownerReference delegation)

3676 i. If opCode = TPM\_FAMILY\_CREATE

3677 (1) The TPM MUST ignore familyID

3678 ii. Else

3679 (1) Verify that the familyID associated with authHandle matches the  
3680 familyID parameter, return TPM\_DELEGATE\_FAMILY on error

36813. Else

3682 a. If TPM\_PERMANENT\_DATA -> ownerAuth is valid, return TPM\_AUTHFAIL

3683 b. If opCode != TPM\_FAMILY\_CREATE and FR -> flags ->  
3684 TPM\_DELEGATE\_ADMIN\_LOCK is TRUE, return TPM\_DELEGATE\_LOCK

3685 c. Validate max NV writes without an owner

3686 i. Set NV1 to TPM\_PERMANENT\_DATA -> noOwnerNVWrite

3687 ii. Increment NV1 by 1

3688 iii. If NV1 > TPM\_MAX\_NV\_WRITE\_NOOWNER return TPM\_MAXNVWRITES

3689 iv. Set TPM\_PERMANENT\_DATA -> noOwnerNVWrite to NV1

36904. The TPM invalidates sessions

3691 a. MUST invalidate all DSAP sessions

3692 b. MUST invalidate all OSAP sessions associated with the delegation table

3693 c. MUST set TPM\_STCLEAR\_DATA -> ownerReference to TPM\_KH\_OWNER

3694 d. MAY invalidate any other session

36955. If opCode == TPM\_FAMILY\_CREATE

3696 a. Validate that sufficient space exists within the TPM to store an additional family and  
3697 map F2 to the newly allocated space.

3698 b. Validate that opData is a TPM\_FAMILY\_LABEL

3699 i. If opDataSize != sizeof(TPM\_FAMILY\_LABEL) return TPM\_BAD\_PARAM\_SIZE

3700 c. Map F2 to a TPM\_FAMILY\_TABLE\_ENTRY

3701 i. Set F2 -> tag to TPM\_TAG\_FAMILY\_TABLE\_ENTRY

3702 ii. Set F2 -> familyLabel to opData

3703 d. Increment TPM\_PERMANENT\_DATA -> lastFamilyID by 1

3704 e. Set F2 -> familyID = TPM\_PERMANENT\_DATA -> lastFamilyID

3705 f. Set F2 -> verificationCount = 1

3706 g. Set F2 -> flags -> TPM\_FAMFLAG\_ENABLED to FALSE

3707 h. Set F2 -> flags -> TPM\_DELEGATE\_ADMIN\_LOCK to FALSE

3708 i. Set retDataSize = 4

3709 j. Set retData = F2 -> familyID  
3710 k. Return TPM\_SUCCESS  
37116. If authHandle is of type DSAP then continueAuthSession MUST set to FALSE  
37127. If opCode == TPM\_FAMILY\_ADMIN  
3713 a. Validate that opDataSize == 1, and that opData is a Boolean value.  
3714 b. Set (FR -> flags -> TPM\_DELEGATE\_ADMIN\_LOCK) = opData  
3715 c. Set retDataSize = 0  
3716 d. Return TPM\_SUCCESS  
37178. else If opCode == TPM\_FAMILY\_ENABLE  
3718 a. Validate that opDataSize == 1, and that opData is a Boolean value.  
3719 b. Set FR -> flags -> TPM\_FAMFLAG\_ENABLED = opData  
3720 c. Set retDataSize = 0  
3721 d. Return TPM\_SUCCESS  
37229. else If opCode == TPM\_FAMILY\_INVALIDATE  
3723 a. Invalidate all data associated with familyRow  
3724 i. All data is all information pointed to by FR  
3725 ii. return TPM\_SELFTEST\_FAILED on failure  
3726 b. The TPM MAY invalidate delegate rows that contain the same familyID.  
3727 c. Set retDataSize = 0  
3728 d. Return TPM\_SUCCESS  
372910. Else return TPM\_BAD\_PARAMETER

## 373019.2 TPM\_Delegate\_CreateKeyDelegation

### 3731Start of informative comment:

3732This command delegates privilege to use a key by creating a blob that can be used by  
3733TPM\_DSAP.

3734There is no check for appropriateness of the key's key usage against the key permission  
3735settings. If the key usage is incorrect, this command succeeds, but the delegated command  
3736will fail.

3737These blobs CANNOT be used as input data for TPM\_LoadOwnerDelegation because the  
3738internal TPM delegate table can store owner delegations only.

3739(TPM\_Delegate\_CreateOwnerDelegation must be used to delegate Owner privilege.)

3740The publicInfo -> familyID can specify a disabled family row. The family row is checked  
3741when the key delegation is used in a DSAP session, not when it is created.

### 3742End of informative comment.

### 3743Incoming 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

3744

### 3745Outgoing 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

### 3746Description

37471. The use restrictions that may be present on the key pointed to by keyHandle are not  
3748 enforced for this command. Stated another way, TPM\_CreateKeyDelegation is not a use  
3749 of the key.

### 3750Action

37511. Verify AuthData for the command and parameters using privAuth
37522. Locate publicInfo -> familyID in the TPM\_FAMILY\_TABLE and set familyRow to indicate  
3753 row, return TPM\_BADINDEX if not found
37543. If the key authentication is in fact a delegation, then the TPM SHALL validate the  
3755 command and parameters using Delegation authorisation, then
  - 3756 a. Validate that authHandle -> familyID equals publicInfo -> familyID return  
3757 TPM\_DELEGATE\_FAMILY on error
  - 3758 b. If TPM\_FAMILY\_TABLE.famTableRow[ authHandle -> familyID] -> flags ->  
3759 TPM\_FAMFLAG\_ENABLED is FALSE, return error TPM\_DISABLED\_CMD.
  - 3760 c. Verify that the delegation bits in publicInfo do not grant more permissions then  
3761 currently delegated. Otherwise return error TPM\_AUTHFAIL
37624. Check that publicInfo -> delegateType is TPM\_DEL\_KEY\_BITS
37635. Verify that authHandle indicates an OSAP or DSAP session return  
3764 TPM\_INVALID\_AUTHHANDLE on error
37656. Create a1 by decrypting delAuth according to the ADIP indicated by authHandle.
37667. Create h1 the SHA-1 of TPM\_STORE\_PUBKEY structure of the key pointed to by  
3767 keyHandle
37688. Create M1 a TPM\_DELEGATE\_SENSITIVE structure
  - 3769 a. Set M1 -> tag to TPM\_TAG\_DELEGATE\_SENSITIVE

- 3770    b. Set M1 -> authValue to a1
- 3771    c. The TPM MAY add additional information of a sensitive nature relative to the  
3772    delegation
- 3773 9. Create M2 the encryption of M1 using TPM\_DELEGATE\_KEY
- 3774 10. Create P1 a TPM\_DELEGATE\_KEY\_BLOB
- 3775    a. Set P1 -> tag to TPM\_TAG\_DELG\_KEY\_BLOB
- 3776    b. Set P1 -> pubKeyDigest to H1
- 3777    c. Set P1 -> pub to PublicInfo
- 3778    d. Set P1 -> pub -> verificationCount to familyRow -> verificationCount
- 3779    e. Set P1 -> integrityDigest to all zeros
- 3780    f. The TPM sets additionalArea and additionalAreaSize appropriate for this TPM. The  
3781    information MAY include symmetric IV, symmetric mode of encryption and other data  
3782    that allows the TPM to process the blob in the future.
- 3783    g. Set P1 -> sensitiveSize to the size of M2
- 3784    h. Set P1 -> sensitiveArea to M2
- 3785 11. Calculate H2 the HMAC of P1 using tpmProof as the secret
- 3786 12. Set P1 -> integrityDigest to H2
- 3787 13. Ignore continueAuthSession on input set continueAuthSession to FALSE on output
- 3788 14. Return P1 as blob

### 3789**19.3 TPM\_Delegate\_CreateOwnerDelegation**

#### 3790**Start of informative comment:**

3791TPM\_Delegate\_CreateOwnerDelegation delegates the Owner's privilege to use a set of  
3792command ordinals, by creating a blob. Such blobs can be used as input data for TPM\_DSAP  
3793or TPM\_Delegate\_LoadOwnerDelegation.

3794TPM\_Delegate\_CreateOwnerDelegation includes the ability to void all existing delegations  
3795(by incrementing the verification count) before creating the new delegation. This ensures  
3796that the new delegation will be the only delegation that can operate at Owner privilege in  
3797this family. This new delegation could be used to enable a security monitor (a local separate  
3798entity, or remote separate entity, or local host entity) to reinitialize a family and perhaps  
3799perform external verification of delegation settings. Normally the ordinals for a delegated  
3800security monitor would include TPM\_Delegate\_CreateOwnerDelegation (this command) in  
3801order to permit the monitor to create further delegations, and  
3802TPM\_Delegate\_UpdateVerification to reactivate some previously voided delegations.

3803If the verification count is incremented and the new delegation does not delegate any  
3804privileges (to any ordinals) at all, or uses an authorisation value that is then discarded, this  
3805family's delegations are all void and delegation must be managed using actual Owner  
3806authorisation.

3807(TPM\_Delegate\_CreateKeyDelegation must be used to delegate privilege to use a key.)

#### 3808**End of informative comment.**

#### 3809**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



**3810 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

**3811 Action**

38121. The TPM SHALL authenticate the command using TPM Owner authentication. Return  
3813 TPM\_AUTHFAIL on failure.

38142. Locate publicInfo -> familyID in the TPM\_FAMILY\_TABLE and set familyRow to indicate  
3815 the row return TPM\_BADINDEX if not found

3816 a. Set FR to TPM\_FAMILY\_TABLE.famTableRow[familyRow]

38173. If the TPM Owner authentication is in fact a delegation

3818 a. Validate that authHandle -> familyID equals publicInfo -> familyID return  
3819 TPM\_DELEGATE\_FAMILY on error

3820 b. If FR -> flags -> TPM\_FAMFLAG\_ENABLED is FALSE, return error  
3821 TPM\_DISABLED\_CMD.

3822 c. Verify that the delegation bits in publicInfo do not grant more permissions then  
3823 currently delegated. Otherwise, return error TPM\_AUTHFAIL.

38244. Check that publicInfo -> delegateType is TPM\_DEL\_OWNER\_BITS

38255. Verify that authHandle indicates an OSAP or DSAP session return  
3826 TPM\_INVALID\_AUTHHANDLE on error

38276. If increment == TRUE

3828 a. Increment FR -> verificationCount

3829 b. Set TPM\_STCLEAR\_DATA-> ownerReference to TPM\_KH\_OWNER

3830 c. The TPM invalidates sessions

3831 i. MUST invalidate all DSAP sessions

3832 ii. MUST invalidate all OSAP sessions associated with the delegation table

3833 iii. MAY invalidate any other session

38347. Create a1 by decrypting delAuth according to the ADIP indicated by authHandle.
38358. Create M1 a TPM\_DELEGATE\_SENSITIVE structure
- 3836 a. Set M1 -> tag to TPM\_TAG\_DELEGATE\_SENSITIVE
- 3837 b. Set M1 -> authValue to a1
- 3838 c. Set other M1 fields as determined by the TPM vendor
38399. Create M2 the encryption of M1 using TPM\_DELEGATE\_KEY
384010. Create B1 a TPM\_DELEGATE\_OWNER\_BLOB
- 3841 a. Set B1 -> tag to TPM\_TAG\_DELG\_OWNER\_BLOB
- 3842 b. Set B1 -> pub to publicInfo
- 3843 c. Set B1 -> sensitiveSize to the size of M2
- 3844 d. Set B1 -> sensitiveArea to M2
- 3845 e. Set B1 -> integrityDigest to all zeros
- 3846 f. Set B1 -> pub -> verificationCount to FR -> verificationCount
384711. The TPM sets additionalArea and additionalAreaSize appropriate for this TPM. The
- 3848 information MAY include symmetric IV, symmetric mode of encryption and other data
- 3849 that allows the TPM to process the blob in the future.
385012. Create H1 the HMAC of B1 using tpmProof as the secret
385113. Set B1 -> integrityDigest to H1
385214. Ignore continueAuthSession on input set continueAuthSession to FALSE on output
385315. Return B1 as blob

3854

19.4 TPM\_Delegate\_LoadOwnerDelegation

3855

Start of informative comment:

3856This command loads a delegate table row blob into a non-volatile delegate table row.  
3857TPM\_Delegate\_LoadOwnerDelegation can be used during manufacturing or on first boot  
3858(when no Owner is installed), or after an Owner is installed. If an Owner is installed,  
3859TPM\_Delegate\_LoadOwnerDelegation requires Owner authorisation, and sensitive  
3860information must be encrypted.

3861Burn-out of TPM non-volatile storage by inappropriate use is mitigated by the TPM’s normal  
3862limits on NV-writes in the absence of an Owner. Tables can be locked after loading using  
3863TPM\_Delegate\_Manage, to prevent subsequent tampering.

3864A management system outside the TPM is expected to manage the delegate table rows  
3865stored on the TPM, and can overwrite any previously stored data. There is no way to  
3866explicitly delete a delegation entry. A new entry can overwrite an invalid entry.

3867This command cannot be used to load key delegation blobs into the TPM

3868

End of informative comment.

3869

Incoming Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_Delegate_LoadOwnerDelegation
4	4	3S	4	TPM_DELEGATE_INDEX	index	The index of the delegate row to be written
4						
3S						
4						
TPM_DELEGATE_INDEX						
index						
The index of the delegate row						

w to be wri tte n						
5	4	4S	4	UINT32	blobSize	The size of the delegate blob
6	<>	5S	<>	TPM_DELEGATE_OWNER_BLOB	blob	Delegation information, including encrypted portions as appropriate
7	4			TPM_AUTHHANDLE	authHandle	The authorization session handle TPM Owner authentication
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
8	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
10	20			TPM_AUTHDATA	ownerAuth	The authorization session digest. HMAC key:ownerAuth

## 3870 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.

## 3871 Actions

38721. Map blob to D1 a TPM\_DELEGATE\_OWNER\_BLOB.

3873 a. Validate that D1 -> tag == TPM\_TAG\_DELEGATE\_OWNER\_BLOB

38742. Locate D1 -> pub -> familyID in the TPM\_FAMILY\_TABLE and set familyRow to indicate  
3875 row, return TPM\_BADINDEX if not found

38763. Set FR to TPM\_FAMILY\_TABLE -> famTableRow[familyRow]

38774. If TPM Owner is installed

3878 a. Validate the command and parameters using TPM Owner authentication, return  
3879 TPM\_AUTHFAIL on error

3880 b. If the command is delegated (authHandle session type is TPM\_PID\_DSAP or through  
3881 ownerReference delegation), verify that D1 -> pub -> familyID matches authHandle ->  
3882 familyID, on error return TPM\_DELEGATE\_FAMILY

38835. Else

3884 a. If tag is not TPM\_TAG\_RQU\_COMMAND, return TPM\_AUTHFAIL

3885 b. If FR -> flags -> TPM\_DELEGATE\_ADMIN\_LOCK is TRUE return  
3886 TPM\_DELEGATE\_LOCK

3887 c. Validate max NV writes without an owner

3888 i. Set NV1 to PD -> noOwnerNVWrite

3889 ii. Increment NV1 by 1

3890 iii. If NV1 > TPM\_MAX\_NV\_WRITE\_NOOWNER return TPM\_MAXNVWRITES

3891 iv. Set PD -> noOwnerNVWrite to NV1

38926. If FR -> flags -> TPM\_FAMFLAG\_ENABLED is FALSE, return TPM\_DISABLED\_CMD

38937. If TPM Owner is installed, validate the integrity of the blob

3894 a. Copy D1 -> integrityDigest to H2

3895 b. Set D1 -> integrityDigest to all zeros

3896 c. Create H3 the HMAC of D1 using tpmProof as the secret  
3897 d. Compare H2 to H3, return TPM\_AUTHFAIL on mismatch  
38988. If TPM Owner is installed, create S1 a TPM\_DELEGATE\_SENSITIVE area by decrypting  
3899 D1 -> sensitiveArea using TPM\_DELEGATE\_KEY. Otherwise set S1 = D1 -> sensitiveArea  
39009. Validate S1  
3901 a. Validate that S1 -> tag == TPM\_TAG\_DELEGATE\_SENSITIVE, return  
3902 TPM\_INVALID\_STRUCTURE on error  
390310. Validate that index is a valid value for delegateTable, return TPM\_BADINDEX on error  
390411. The TPM invalidates sessions  
3905 a. MUST invalidate all DSAP sessions  
3906 b. MUST invalidate all OSAP sessions associated with the delegation table  
3907 c. MAY invalidate any other session  
390812. Copy data to the delegate table row  
3909 a. Copy the TPM\_DELEGATE\_PUBLIC from D1 -> pub to TPM\_DELEGATE\_TABLE ->  
3910 delRow[index] -> pub.  
3911 b. Copy the TPM\_SECRET from S1 -> authValue to TPM\_DELEGATE\_TABLE ->  
3912 delRow[index] -> authValue.  
3913 c. Set TPM\_STCLEAR\_DATA -> ownerReference to TPM\_KH\_OWNER  
3914 d. If authHandle is of type DSAP then continueAuthSession MUST set to FALSE  
391513. Return TPM\_SUCCESS

## 391619.5 TPM\_Delegate\_ReadTable

### 3917Start of informative comment:

3918This command reads from the TPM the public contents of the family and delegate tables  
3919that are stored on the TPM. Such data is required during external verification of tables.

3920There are no restrictions on the execution of this command; anyone can read this  
3921information regardless of the state of the PCRs, regardless of whether they know any  
3922specific AuthData value and regardless of whether or not the enable and admin bits are set  
3923one way or the other.

### 3924End of informative comment.

### 3925Incoming 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

### 3926Outgoing 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

### 3927Actions

39281. Set familyTableSize to the number of valid families on the TPM times  
3929 sizeof(TPM\_FAMILY\_TABLE\_ELEMENT).

39302. Copy the valid entries in the internal family table to the output array familyTable

39313. Set delegateTableSize to the number of valid delegate table entries on the TPM times  
3932 (sizeof(TPM\_DELEGATE\_PUBLIC) + 4).

39334. For each valid entry

3934 a. Write the TPM\_DELEGATE\_INDEX to delegateTable

3935 b. Copy the TPM\_DELEGATE\_PUBLIC to delegateTable

39365. Return TPM\_SUCCESS



## 3937**19.6 TPM\_Delegate\_UpdateVerification**

### 3938**Start of informative comment:**

3939TPM\_UpdateVerification sets the verificationCount in an entity (a blob or a delegation row)  
3940to the current family value, in order that the delegations represented by that entity will  
3941continue to be accepted by the TPM.

### 3942**End of informative comment.**

### 3943**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.

### 3944Outgoing 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.

### 3945Actions

39461. Verify the TPM Owner, directly or indirectly through delegation, authorizes the command  
3947 and parameters, on error return TPM\_AUTHFAIL
39482. Determine the type of inputData (TPM\_DELEGATE\_TABLE\_ROW or  
3949 TPM\_DELEGATE\_OWNER\_BLOB or TPM\_DELEGATE\_KEY\_BLOB) and map D1 to that  
3950 structure
- 3951 a. Mapping to TPM\_DELEGATE\_TABLE\_ROW requires taking inputData as a tableIndex  
3952 and locating the appropriate row in the table
39533. If D1 is a TPM\_DELEGATE\_OWNER\_BLOB or TPM\_DELEGATE\_KEY\_BLOB, validate the  
3954 integrity of D1
- 3955 a. Copy D1 -> integrityDigest to H2
- 3956 b. Set D1 -> integrityDigest to all zeros
- 3957 c. Create H3 the HMAC of D1 using tpmProof as the secret
- 3958 d. Compare H2 to H3 return TPM\_AUTHFAIL on mismatch
39594. Locate (D1 -> pub -> familyID) in the TPM\_FAMILY\_TABLE and set familyRow to indicate  
3960 row, return TPM\_BADINDEX if not found
39615. Set FR to TPM\_FAMILY\_TABLE.famTableRow[familyRow]
39626. If delegated, verify that family of the delegated Owner-auth is the same as D1:  
3963 (authHandle -> familyID) == (D1 -> pub -> familyID); otherwise return error  
3964 TPM\_DELEGATE\_FAMILY
39657. If delegated, verify that the family of the delegated Owner-auth is enabled: if (authHandle  
3966 -> familyID -> flags TPM\_FAMFLAG\_ENABLED) is FALSE, return TPM\_DISABLED\_CMD
39678. Set D1 -> verificationCount to FR -> verificationCount

39689. If D1 is a TPM\_DELEGATE\_OWNER\_BLOB or TPM\_DELEGATE\_KEY\_BLOB set the  
3969 integrity of D1

3970 a. Set D1 -> integrityDigest to all zeros

3971 b. Create H1 the HMAC of D1 using tpmProof as the secret

3972 c. Set D1 -> integrityDigest to H1

397310. If D1 is a blob recreate the blob and return it

## 397419.7 TPM\_Delegate\_VerifyDelegation

### 3975Start of informative comment:

3976TPM\_VerifyDelegation interprets a delegate blob and returns success or failure, depending  
3977on whether the blob is currently valid. The delegate blob is NOT loaded into the TPM.

### 3978End of informative comment.

### 3979Incoming 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

### 3980Outgoing 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

### 3981Actions

39821. Determine the type of blob, If delegation -> tag is equal to  
3983 TPM\_TAG\_DELGATE\_OWNER\_BLOB then

3984 a. Map D1 a TPM\_DELEGATE\_OWNER\_BLOB to delegation

39852. Else if delegation -> tag = TPM\_TAG\_DELG\_KEY\_BLOB

3986 a. Map D1 a TPM\_DELEGATE\_KEY\_BLOB to delegation

39873. Else return TPM\_BAD\_PARAMETER

39884. Locate D1 -> familyID in the TPM\_FAMILY\_TABLE and set familyRow to indicate row,  
3989 return TPM\_BADINDEX if not found

39905. Set FR to TPM\_FAMILY\_TABLE.famTableRow[familyRow]

39916. If FR -> flags TPM\_FAMFLAG\_ENABLED is FALSE, return TPM\_DISABLED\_CMD

39927. Validate that D1 -> pub -> verificationCount matches FR -> verificationCount, on  
3993 mismatch return TPM\_FAMILYCOUNT

39948. Validate the integrity of D1

3995 a. Copy D1 -> integrityDigest to H2

991

992

3996 b. Set D1 -&gt; integrityDigest to all zeros

3997 c. Create H3 the HMAC of D1 using tpmProof as the secret

3998 d. Compare H2 to H3 return TPM\_AUTHFAIL on mismatch

39999. Create S1 a TPM\_DELEGATE\_SENSITIVE area by decrypting D1 -&gt; sensitiveArea using

4000 TPM\_DELEGATE\_KEY

400110. Validate S1 values

4002 a. S1 -&gt; tag is TPM\_TAG\_DELEGATE\_SENSITIVE

4003 b. Return TPM\_BAD\_PARAMETER on error

400411. Return TPM\_SUCCESS

## 4005**20. Non-volatile Storage**

4006**Start of informative comment:**

4007This section handles the allocation and use of the TPM non-volatile storage.

4008**End of informative comment.**

4009If nvIndex refers to the DIR, the TPM ignores actions containing access control checks that  
4010have no meaning for the DIR. The TPM only checks the owner authorization.

## 401220.1 TPM\_NV\_DefineSpace

### 4013Start of informative comment:

4014This establishes the space necessary for the indicated index. The definition will include the  
4015access requirements for writing and reading the area.

4016 Previously defined space at the index and new size is non-zero (and space is available,  
4017 etc.) -> redefine the index

4018 No previous space at the index and new size is non-zero (and space is available,  
4019 etc.)-> define the index

4020 Previously defined space at the index and new size is 0 -> delete the index

4021 No previous space at the index and new size is 0 -> error

4022The space definition size does not include the area needed to manage the space.

4023Setting TPM\_PERMANENT\_FLAGS -> nvLocked TRUE when it is already TRUE is not an  
4024error.

### 4025End of informative comment.

### 4026Incoming 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

### 4027Outgoing 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

## 4028Description

4029For the case where pubInfo -> dataSize is 0, pubInfo -> pcrInfoRead and pubInfo ->  
4030pcrInfoWrite are not used. However, since the general principle is to validate parameters  
4031before changing state, the TPM SHOULD parse pubInfo completely before invalidating the  
4032data area.

## 4033Actions

40341. If pubInfo -> nvIndex == TPM\_NV\_INDEX\_LOCK and tag = TPM\_TAG\_RQU\_COMMAND
  - 4035 a. If pubInfo -> dataSize is not 0, the command MAY return TPM\_BADINDEX.
  - 4036 b. Set TPM\_PERMANENT\_FLAGS -> nvLocked to TRUE
  - 4037 c. Return TPM\_SUCCESS
40382. If TPM\_PERMANENT\_FLAGS -> nvLocked is FALSE then all authorization checks except  
4039 for the Max NV writes are ignored
  - 4040 a. Ignored checks include physical presence, owner authorization, 'D' bit check,  
4041 bGlobalLock, no authorization with a TPM owner present, bWriteSTClear, the check that  
4042 pubInfo -> dataSize is 0 in Action 5.c. (the no-authorization case) , disabled and  
4043 deactivated.
    - 4044 i. The check that pubInfo -> dataSize is 0 is still enforced in Action 6.f.  
4045 (returning after deleting a previously defined storage area) and Action 9.f. (not  
4046 allowing a space of size 0 to be defined).
    - 4047 ii. If ownerAuth is present, the TPM MAY check the authorization HMAC.
  - 4048 b. The check for pubInfo -> nvIndex == TPM\_NV\_INDEX0 in Action 3. is not ignored.
40493. If pubInfo -> nvIndex has the D bit (bit 28) set to a 1 or pubInfo -> nvIndex ==  
4050 TPM\_NV\_INDEX0 then
  - 4051 a. Return TPM\_BADINDEX
  - 4052 b. The D bit specifies an index value that is set in manufacturing and can never be  
4053 deleted or added to the TPM
  - 4054 c. Index value TPM\_NV\_INDEX0 is reserved and cannot be defined
40554. If tag = TPM\_TAG\_RQU\_AUTH1\_COMMAND then
  - 4056 a. The TPM MUST validate the command and parameters using the TPM Owner  
4057 authentication and ownerAuth, on error return TPM\_AUTHFAIL
  - 4058 b. authHandle session type MUST be OSAP
  - 4059 c. Create A1 by decrypting encAuth according to the ADIP indicated by authHandle.
40605. else
  - 4061 a. Validate the assertion of physical presence. Return TPM\_BAD\_PRESENCE on error.



- 1009  
1010
- 4062 b. If TPM Owner is present then return TPM\_OWNER\_SET.
- 4063 c. If pubInfo -> dataSize is 0 then return TPM\_BAD\_DATASIZE. Setting the size to 0  
4064 represents an attempt to delete the value without TPM Owner authentication.
- 4065 d. Validate max NV writes without an owner
- 4066 i. Set NV1 to TPM\_PERMANENT\_DATA -> noOwnerNVWrite
- 4067 ii. Increment NV1 by 1
- 4068 iii. If NV1 > TPM\_MAX\_NV\_WRITE\_NOOWNER return TPM\_MAXNVWRITES
- 4069 iv. Set NV1\_INCREMENTED to TRUE
- 4070 e. Set A1 to encAuth. There is no nonce or authorization to create the encryption string,  
4071 hence the AuthData value is passed in the clear
40726. If pubInfo -> nvIndex points to a valid previously defined storage area then
- 4073 a. Map D1 a TPM\_NV\_DATA\_SENSITIVE to the storage area
- 4074 b. If D1 -> attributes specifies TPM\_NV\_PER\_GLOBALLOCK then
- 4075 i. If TPM\_STCLEAR\_FLAGS -> bGlobalLock is TRUE then return  
4076 TPM\_AREA\_LOCKED
- 4077 c. If D1 -> attributes specifies TPM\_NV\_PER\_WRITE\_STCLEAR
- 4078 i. If D1 -> pubInfo -> bWriteSTClear is TRUE then return TPM\_AREA\_LOCKED
- 4079 d. Invalidate the data area currently pointed to by D1 and ensure that if the area is  
4080 reallocated no residual information is left
- 4081 e. If NV1\_INCREMENTED is TRUE
- 4082 i. Set TPM\_PERMANENT\_DATA -> noOwnerNVWrite to NV1
- 4083 f. The TPM invalidates authorization sessions
- 4084 i. MUST invalidate all authorization sessions associated with D1
- 4085 ii. MAY invalidate any other authorization session
- 4086 g. If pubInfo -> dataSize is 0 then return TPM\_SUCCESS
40877. Parse pubInfo -> pcrInfoRead
- 4088 a. Validate pcrInfoRead structure on error return TPM\_INVALID\_STRUCTURE
- 4089 i. Validation includes proper PCR selections and locality selections
40908. Parse pubInfo -> pcrInfoWrite
- 4091 a. Validate pcrInfoWrite structure on error return TPM\_INVALID\_STRUCTURE
- 4092 i. Validation includes proper PCR selections and locality selections
- 4093 b. If pcrInfoWrite -> localityAtRelease disallows some localities
- 4094 i. Set writeLocalities to TRUE
- 4095 c. Else
- 4096 i. Set writeLocalities to FALSE

40979. Validate that the attributes are consistent
- 4098 a. The TPM SHALL ignore the bReadSTClear, bWriteSTClear and bWriteDefine  
4099 attributes during the execution of this command
- 4100 b. If TPM\_NV\_PER\_OWNERWRITE is TRUE and TPM\_NV\_PER\_AUTHWRITE is TRUE  
4101 return TPM\_AUTH\_CONFLICT
- 4102 c. If TPM\_NV\_PER\_OWNERREAD is TRUE and TPM\_NV\_PER\_AUTHREAD is TRUE  
4103 return TPM\_AUTH\_CONFLICT
- 4104 d. If TPM\_NV\_PER\_OWNERWRITE and TPM\_NV\_PER\_AUTHWRITE and  
4105 TPM\_NV\_PER\_WRITEDEFINE and TPM\_NV\_PER\_PPWRITE and writeLocalities are all  
4106 FALSE
- 4107 i. Return TPM\_PER\_NOWRITE
- 4108 e. Validate pubInfo -> nvIndex
- 4109 i. Make sure that the index is applicable for this TPM. Return TPM\_BADINDEX  
4110 on error. A valid index is platform and context sensitive. That is, attempting to  
4111 validate an index may be successful in one configuration and invalid in another  
4112 configuration. The individual index values MUST indicate if there are any  
4113 restrictions on the use of the index.
- 4114 ii. TPM\_NV\_INDEX\_DIR is always an invalid defined index.
- 4115 f. If dataSize is 0 return TPM\_BAD\_PARAM\_SIZE
411610. Create D1 a TPM\_NV\_DATA\_SENSITIVE structure
- 4117 a. Set D1 -> pubInfo to pubInfo
- 4118 b. Set D1 -> authValue to A1
- 4119 c. Set D1 -> pubInfo -> bReadSTClear to FALSE
- 4120 d. Set D1 -> pubInfo -> bWriteSTClear to FALSE
- 4121 e. Set D1 -> pubInfo -> bWriteDefine to FALSE
412211. Validate that sufficient NV is available to store D1 and pubInfo -> dataSize bytes of data
- 4123 a. Return TPM\_NOSPACE if pubInfo -> dataSize is not available in the TPM
412412. If pubInfo -> nvIndex is not TPM\_NV\_INDEX\_TRIAL
- 4125 a. Reserve NV space for pubInfo -> dataSize
- 4126 b. Set all bytes in the newly defined area to 0xFF
- 4127 c. If NV1\_INCREMENTED is TRUE
- 4128 i. Set TPM\_PERMANENT\_DATA -> noOwnerNVWrite to NV1
412913. Ignore continueAuthSession on input and set to FALSE on output
413014. Return TPM\_SUCCESS

## 4131**20.2 TPM\_NV\_WriteValue**

### 4132**Start of informative comment:**

4133This command writes the value to a defined area. The write can be TPM Owner authorized  
4134or unauthorized and protected by other attributes and will work when no TPM Owner is  
4135present.

4136The action setting bGlobalLock to TRUE is intentionally before the action checking the  
4137owner authorization. This allows code (e.g., a BIOS) to lock NVRAM without knowing the  
4138owner authorization.

4139The DIR (TPM\_NV\_INDEX\_DIR) has the attributes TPM\_NV\_PER\_OWNERWRITE and  
4140TPM\_NV\_WRITEALL.

4141Certain platform manufacturers or software might require specific error handling in Action  
414220.2.

4143Owner authorization is not required when nvLocked is FALSE. If the host does send owner  
4144authorization, Action 20.2 indicates that it should be correct, since some TPM  
4145implementations may validate it.

### 4146**End of informative comment.**

### 4147**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

### 4148**Outgoing Operands and Sizes**

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

4	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	authNonceOdd	Nonce generated by caller
5	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
6	20			TPM_AUTHDATA	ownerAuth	The authorization session digest HMAC key: ownerAuth

#### 4149Description

4150For TPM\_NV\_INDEX\_DIR, the ordinal MUST NOT set an error code for the “if dataSize = 0”  
4151action. However, the flags set in this case are not applicable to the DIR.

#### 4152Actions

41531. If TPM\_PERMANENT\_FLAGS -> nvLocked is FALSE then all authorization checks except  
4154 for the max NV writes are ignored

4155 a. Ignored checks include physical presence, owner authorization,  
4156 TPM\_NV\_PER\_OWNERWRITE, PCR, bWriteDefine, bGlobalLock, bWriteSTClear, locality,  
4157 disabled and deactivated.

4158 b. TPM\_NV\_PER\_AUTHWRITE is not ignored.

4159 c. If ownerAuth is present, the TPM MAY check the authorization HMAC.

41602. Locate and set D1 to the TPM\_NV\_DATA\_AREA that corresponds to nvIndex, return  
4161 TPM\_BADINDEX on error

4162 a. If nvIndex = TPM\_NV\_INDEX\_DIR, set D1 to TPM\_PERMANENT\_DATA -> authDir[0]

41633. If TPM\_PERMANENT\_FLAGS -> nvLocked is TRUE

4164 a. If D1 -> permission -> TPM\_NV\_PER\_OWNERWRITE is TRUE

4165 i. If TPM\_PERMANENT\_FLAGS -> disable is TRUE, return TPM\_DISABLED

4166 ii. If TPM\_STCLEAR\_FLAGS -> deactivated is TRUE, return TPM\_DEACTIVATED

4167 b. If D1 -> permission -> TPM\_NV\_PER\_OWNERWRITE is FALSE

4168 i. If TPM\_PERMANENT\_FLAGS -> disable is TRUE, the TPM MAY return  
4169 TPM\_DISABLED

4170 ii. If TPM\_STCLEAR\_FLAGS -> deactivated is TRUE, the TPM MAY return  
4171 TPM\_DEACTIVATED

41724. If tag = TPM\_TAG\_RQU\_AUTH1\_COMMAND then

4173 a. If D1 -> permission -> TPM\_NV\_PER\_OWNERWRITE is FALSE return  
4174 TPM\_AUTH\_CONFLICT

4175 i. This check is ignored if nvIndex is TPM\_NV\_INDEX0.

4176 b. Validate command and parameters using ownerAuth HMAC with TPM Owner  
4177 authentication as the secret, return TPM\_AUTHFAIL on error

41785. Else

4179 a. If D1 -> permission -> TPM\_NV\_PER\_OWNERWRITE is TRUE return  
4180 TPM\_AUTH\_CONFLICT

4181 i. This check is ignored if nvIndex is TPM\_NV\_INDEX0.

1027  
1028

4182 b. If no TPM Owner validate max NV writes without an owner

4183 i. Set NV1 to TPM\_PERMANENT\_DATA -> noOwnerNVWrite

4184 ii. Increment NV1 by 1

4185 iii. If NV1 > TPM\_MAX\_NV\_WRITE\_NOOWNER return TPM\_MAXNVWRITES

4186 iv. Set NV1\_INCREMENTED to TRUE

4187 6. If nvIndex is TPM\_NV\_INDEX0 then

4188 a. If dataSize is not 0, the TPM MAY return TPM\_BADINDEX.

4189 b. Set TPM\_STCLEAR\_FLAGS -> bGlobalLock to TRUE

4190 c. Return TPM\_SUCCESS

4191 7. If D1 -> permission -> TPM\_NV\_PER\_AUTHWRITE is TRUE return

4192 TPM\_AUTH\_CONFLICT

4193 8. Check that D1 -> pcrInfoWrite -> localityAtRelease for TPM\_STANY\_DATA ->

4194 localityModifier is TRUE

4195 a. For example if TPM\_STANY\_DATA -> localityModifier was 2 then D1 -> pcrInfo ->

4196 localityAtRelease -> TPM\_LOC\_TWO would have to be TRUE

4197 b. On error return TPM\_BAD\_LOCALITY

4198 9. If D1 -> attributes specifies TPM\_NV\_PER\_PPWRITE then validate physical presence is

4199 asserted if not return TPM\_BAD\_PRESENCE

4200 10. If D1 -> attributes specifies TPM\_NV\_PER\_WRITEDEFINE

4201 a. If D1 -> bWriteDefine is TRUE return TPM\_AREA\_LOCKED

4202 11. If D1 -> attributes specifies TPM\_NV\_PER\_GLOBALLOCK

4203 a. If TPM\_STCLEAR\_FLAGS -> bGlobalLock is TRUE return TPM\_AREA\_LOCKED

4204 12. If D1 -> attributes specifies TPM\_NV\_PER\_WRITE\_STCLEAR

4205 a. If D1 -> bWriteSTClear is TRUE return TPM\_AREA\_LOCKED

4206 13. If D1 -> pcrInfoWrite -> pcrSelection specifies a selection of TPM\_STCLEAR\_DATA ->

4207 PCR[]

4208 a. Create P1 a composite hash of the TPM\_STCLEAR\_DATA -> PCR[] specified by D1 ->

4209 pcrInfoWrite

4210 b. Compare P1 to D1 -> pcrInfoWrite -> digestAtRelease return TPM\_WRONGPCRVAL

4211 on mismatch

4212 14. If dataSize = 0 then

4213 a. Set D1 -> bWriteSTClear to TRUE

4214 b. Set D1 -> bWriteDefine to TRUE

4215 15. Else

4216 a. Set S1 to offset + dataSize

4217 b. If S1 > D1 -> dataSize return TPM\_NOSPACE

4218 c. If D1 -> attributes specifies TPM\_NV\_PER\_WRITEALL  
4219 i. If dataSize != D1 -> dataSize return TPM\_NOT\_FULLWRITE  
4220 d. Write the new value into the NV storage area  
4221 e. If NV1\_INCREMENTED is TRUE  
4222 i. Set TPM\_PERMANENT\_DATA -> noOwnerNVWrite to NV1  
4223 16. Set D1 -> bReadSTClear to FALSE  
4224 17. Return TPM\_SUCCESS

## 422520.3 TPM\_NV\_WriteValueAuth

### 4226Start of informative comment:

4227This command writes to a previously defined area. The area must require authorization to  
4228write. Use this command when authorization other than the owner authorization is to be  
4229used. Otherwise, use TPM\_NV\_WriteValue.

4230The Part 2 ordinal table indicates that TPM\_NV\_WriteValueAuth requires an owner present.  
4231This is normative, although it was a mistake.

### 4232End of informative comment.

### 4233Incoming 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

### 4234Outgoing 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

### 4235Actions

42361. Locate and set D1 to the TPM\_NV\_DATA\_AREA that corresponds to nvIndex, return  
4237 TPM\_BADINDEX on error

42382. If D1 -> attributes does not specify TPM\_NV\_PER\_AUTHWRITE then return  
4239 TPM\_AUTH\_CONFLICT

42403. Validate authValue using D1 -> authValue, return TPM\_AUTHFAIL on error

42414. Check that D1 -> pcrInfoWrite -> localityAtRelease for TPM\_STANY\_DATA ->  
4242 localityModifier is TRUE

4243 a. For example if TPM\_STANY\_DATA -> localityModifier was 2 then D1 -> pcrInfo ->  
4244 localityAtRelease -> TPM\_LOC\_TWO would have to be TRUE

4245 b. On error return TPM\_BAD\_LOCALITY

42465. If D1 -> attributes specifies TPM\_NV\_PER\_PPWRITE then validate physical presence is  
4247 asserted if not return TPM\_BAD\_PRESENCE

42486. If D1 -> pcrInfoWrite -> pcrSelection specifies a selection of PCR

4249 a. Create P1 a composite hash of the TPM\_STCLEAR\_DATA -> PCR[] specified by D1 ->  
4250 pcrInfoWrite

4251 b. Compare P1 to digestAtRelease return TPM\_WRONGPCRVAL on mismatch

42527. If D1 -> attributes specifies TPM\_NV\_PER\_WRITEDEFINE

4253 a. If D1 -> bWriteDefine is TRUE return TPM\_AREA\_LOCKED

42548. If D1 -> attributes specifies TPM\_NV\_PER\_GLOBALLOCK

4255 a. If TPM\_STCLEAR\_FLAGS -> bGlobalLock is TRUE return TPM\_AREA\_LOCKED

42569. If D1 -> attributes specifies TPM\_NV\_PER\_WRITE\_STCLEAR

4257 a. If D1 -> bWriteSTClear is TRUE return TPM\_AREA\_LOCKED

425810.If dataSize = 0 then

4259 a. Set D1 -> bWriteSTClear to TRUE

4260 b. Set D1 -> bWriteDefine to TRUE

426111.Else

4262 a. Set S1 to offset + dataSize

4263 b. If S1 > D1 -> dataSize return TPM\_NOSPACE

4264 c. If D1 -> attributes specifies TPM\_NV\_PER\_WRITEALL

4265 i. If dataSize != D1 -> dataSize return TPM\_NOT\_FULLWRITE

4266 d. Write the new value into the NV storage area

426712.Set D1 -> bReadSTClear to FALSE

426813.Return TPM\_SUCCESS



## 4269**20.4 TPM\_NV\_ReadValue**

### 4270**Start of informative comment:**

4271Read a value from the NV store. This command uses optional owner authentication.

4272Action 1 indicates that if the NV area is not locked then reading of the NV area continues  
4273without ANY authorization. This is intentional, and allows a platform manufacturer to set  
4274the NV areas, read them back, and then lock them all without having to install a TPM  
4275owner.

4276Certain platform manufacturers or software might require specific error handling in Action  
427720.4.

4278Owner authorization is not required when nvLocked is FALSE. If the host does send owner  
4279authorization, Action 20.4 indicates that it should be correct, since some TPM  
4280implementations may validate it.

### 4281**End of informative comment.**

### 4282**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

### 4283**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

## 4284Actions

42851. If TPM\_PERMANENT\_FLAGS -> nvLocked is FALSE then all authorization checks are  
4286 ignored.
- 4287 a. Ignored checks include physical presence, owner authorization, PCR, bReadSTClear,  
4288 locality, TPM\_NV\_PER\_OWNERREAD, disabled and deactivated.
- 4289 b. TPM\_NV\_PER\_AUTHREAD is not ignored.
- 4290 c. If ownerAuth is present, the TPM MAY check the authorization HMAC.
42912. Set D1 a TPM\_NV\_DATA\_AREA structure to the area pointed to by nvIndex, if not found  
4292 return TPM\_BADINDEX
- 4293 a. If nvIndex = TPM\_NV\_INDEX\_DIR, set D1 to TPM\_PERMANENT\_DATA -> authDir[0]
42943. If TPM\_PERMANENT\_FLAGS -> nvLocked is TRUE
- 4295 a. If D1 -> permission -> TPM\_NV\_PER\_OWNERREAD is TRUE
- 4296 i. If TPM\_PERMANENT\_FLAGS -> disable is TRUE, return TPM\_DISABLED
- 4297 ii. If TPM\_STCLEAR\_FLAGS -> deactivated is TRUE, return TPM\_DEACTIVATED
- 4298 b. If D1 -> permission -> TPM\_NV\_PER\_OWNERREAD is FALSE
- 4299 i. If TPM\_PERMANENT\_FLAGS -> disable is TRUE, the TPM MAY return  
4300 TPM\_DISABLED
- 4301 ii. If TPM\_STCLEAR\_FLAGS -> deactivated is TRUE, the TPM MAY return  
4302 TPM\_DEACTIVATED
43034. If tag = TPM\_TAG\_RQU\_AUTH1\_COMMAND then
- 4304 a. If D1 -> TPM\_NV\_PER\_OWNERREAD is FALSE return TPM\_AUTH\_CONFLICT
- 4305 b. Validate command and parameters using TPM Owners authentication on error return  
4306 TPM\_AUTHFAIL
43075. Else
- 4308 a. If D1 -> TPM\_NV\_PER\_AUTHREAD is TRUE return TPM\_AUTH\_CONFLICT
- 4309 b. If D1 -> TPM\_NV\_PER\_OWNERREAD is TRUE return TPM\_AUTH\_CONFLICT
43106. Check that D1 -> pcrInfoRead -> localityAtRelease for TPM\_STANY\_DATA ->  
4311 localityModifier is TRUE
- 4312 a. For example if TPM\_STANY\_DATA -> localityModifier was 2 then D1 -> pcrInfo ->  
4313 localityAtRelease -> TPM\_LOC\_TWO would have to be TRUE
- 4314 b. On error return TPM\_BAD\_LOCALITY
43157. If D1 -> attributes specifies TPM\_NV\_PER\_PPREAD then validate physical presence is  
4316 asserted if not return TPM\_BAD\_PRESENCE
43178. If D1 -> TPM\_NV\_PER\_READ\_STCLEAR then

1054

1055

4318 a. If D1 -> bReadSTClear is TRUE return TPM\_DISABLED\_CMD

43199. If D1 -> pcrInfoRead -> pcrSelection specifies a selection of PCR

4320 a. Create P1 a composite hash of the TPM\_STCLEAR\_DATA -> PCR[] specified by D1 ->  
4321 pcrInfoRead

4322 b. Compare P1 to D1 -> pcrInfoRead -> digestAtRelease return TPM\_WRONGPCRVAL on  
4323 mismatch

432410. If dataSize is 0 then

4325 a. Set D1 -> bReadSTClear to TRUE

4326 b. Set data to NULL (output parameter dataSize to 0)

432711. Else

4328 a. Set S1 to offset + dataSize

4329 b. If S1 > D1 -> dataSize return TPM\_NOSPACE

4330 c. Set data to area pointed to by offset

4331 i. This includes partial reads of TPM\_NV\_INDEX\_DIR.

433212. Return TPM\_SUCCESS

## 433320.5 TPM\_NV\_ReadValueAuth

### 4334Start of informative comment:

4335This command requires that the read be authorized by a value set with the blob.

4336The Part 2 ordinal table indicates that TPM\_NV\_ReadValueAuth requires an owner present.

4337This is normative, although it was a mistake.

### 4338End of informative comment.

### 4339Incoming 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

### 4340Outgoing 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

### 4341Actions

43421. Locate and set D1 to the TPM\_NV\_DATA\_AREA that corresponds to nvIndex, on error

4343 return TPM\_BADINDEX

43442. If D1 -> TPM\_NV\_PER\_AUTHREAD is FALSE return TPM\_AUTH\_CONFLICT
43453. Validate authHmac using D1 -> authValue on error return TPM\_AUTHFAIL
43464. If D1 -> attributes specifies TPM\_NV\_PER\_PPREAD then validate physical presence is asserted if not return TPM\_BAD\_PRESENCE
43485. Check that D1 -> pcrInfoRead -> localityAtRelease for TPM\_STANY\_DATA -> localityModifier is TRUE
- 4350 a. For example if TPM\_STANY\_DATA -> localityModifier was 2 then D1 -> pcrInfo -> localityAtRelease -> TPM\_LOC\_TWO would have to be TRUE
- 4351
- 4352 b. On error return TPM\_BAD\_LOCALITY
43536. If D1 -> pcrInfoRead -> pcrSelection specifies a selection of PCR
- 4354 a. Create P1 a composite hash of the TPM\_STCLEAR\_DATA -> PCR[] specified by D1 -> pcrInfoRead
- 4355
- 4356 b. Compare P1 to D1 -> pcrInfoRead -> digestAtRelease return TPM\_WRONGPCRVAL on mismatch
- 4357
43587. If D1 specifies TPM\_NV\_PER\_READ\_STCLEAR then
- 4359 a. If D1 -> bReadSTClear is TRUE return TPM\_DISABLED\_CMD
43608. If dataSize is 0 then
- 4361 a. Set D1 -> bReadSTClear to TRUE
- 4362 b. Set data to all zeros
43639. Else
- 4364 a. Set S1 to offset + dataSize
- 4365 b. If S1 > D1 -> dataSize return TPM\_NOSPACE
- 4366 c. Set data to area pointed to by offset
436710. Return TPM\_SUCCESS

## 4368**21. Session Management**

### 4369**Start of informative comment:**

4370Three TPM\_RT\_CONTEXT session resources located in TPM\_STANY\_DATA work together to  
4371control session save and load: contextNonceSession, contextCount, and contextList[].

4372All three MUST be initialized at TPM\_Startup(ST\_CLEAR) and TPM\_Startup(ST\_DEACTIVATED)  
4373and MAY be initialized at TPM\_Startup(ST\_STATE). Initializing invalidates all saved  
4374sessions. They MAY be restored by TPM\_Startup(ST\_STATE). This case would allow saved  
4375sessions to be loaded. The actual ST\_STATE operation is reported by the  
4376TPM\_RT\_CONTEXT startup effect.

4377TPM\_SaveContext creates a contextBlob containing an encrypted contextNonceSession. The  
4378nonce is checked by TPM\_LoadContext. So initializing contextNonceSession invalidates all  
4379saved contexts. The nonce is large and protected, making a replay infeasible.

4380The contextBlob also contains a public but protected contextCount. The count increments  
4381for each saved contextBlob. The TPM also saves contextCount in contextList[]. The TPM  
4382validates contextBlob against the contextList[] during TPM\_LoadContext. Since the  
4383contextList[] is finite, it limits the number of valid saved sessions. Since the contextCount  
4384cannot be allowed to wrap, it limits the total number of saved sessions.

4385After a contextBlob is loaded, its contextCount entry is removed from contextList[]. This  
4386releases space in the context list for future entries. It also invalidates the contextBlob. So a  
4387saved contextBlob can be loaded only once.

4388TPM\_FlushSpecific can also specify a contextCount to be removed from the contextList[],  
4389allowing invalidation of an individual contextBlob. This is different from TPM\_FlushSpecific  
4390specifying a session handle, which invalidates a loaded session, not a saved contextBlob.

### 4391**End of informative comment.**

4392

## 4393**21.1 TPM\_KeyControlOwner**

### 4394**Start of informative comment:**

4395This command controls some attributes of keys that are stored within the TPM key cache.

4396OwnerEvict: If this bit is set to true, this key remains in the TPM non-volatile storage  
4397through all TPM\_Startup events. The only way to evict this key is for the TPM Owner to  
4398execute this command again, setting the owner control bit to false and then executing  
4399TPM\_FlushSpecific.

4400The key handle does not reference an authorized entity and is not validated.

4401The check for two remaining key slots ensures that users can load the two keys required to  
4402execute many commands. Since only the owner can flush owner evict keys, non-owner  
4403commands could be blocked if this test was not performed.

### 4404**End of informative comment.**

## 4405**Incoming Parameters and Sizes**

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

#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes incl. paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_KeyControlOwner
4	4			TPM_KEY_HANDLE	keyHandle	The handle of a loaded key.
5	<>	2S	<>	TPM_PUBKEY	pubKey	The public key associated with the loaded key
6	4	3S	4	TPM_KEY_CONTROL	bitName	The name of the bit to be modified
7	1	4S	1	BOOL	bitValue	The value to set the bit to
8	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for owner authentication.
9		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
10	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
11	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
12	20		20	TPM_AUTHDATA	ownerAuth	HMAC authorization: key ownerAuth

## 4406 Outgoing Parameters and Sizes

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

## 4407 Description

44081. Set an internal bit within the key cache that controls some attribute of a loaded key.

## 4409 Actions

44101. Validate the AuthData using the owner authentication value, on error return  
4411 TPM\_AUTHFAIL

44122. Validate that keyHandle refers to a loaded key, return TPM\_INVALID\_KEYHANDLE on  
4413 error.

44143. Validate that pubKey matches the key held by the TPM pointed to by keyHandle, return  
4415 TPM\_BAD\_PARAMETER on mismatch

4416 a. This check is added so that virtualization of the keyHandle does not result in attacks,  
4417 as the keyHandle is not associated with an authorization value

44184. Validate that bitName is valid, return TPM\_BAD\_MODE on error.

44195. If bitName == TPM\_KEY\_CONTROL\_OWNER\_EVICT

4420     a. If bitValue == TRUE  
4421         i. Verify that after this operation at least two key slots will be present within the  
4422         TPM that can store any type of key both of which do NOT have the OwnerEvict bit  
4423         set, on error return TPM\_NOSPACE  
4424         ii. Verify that for this key handle, parentPCRStatus is FALSE and isVolatile is  
4425         FALSE. Return TPM\_BAD\_PARAMETER on error.  
4426         iii. Set ownerEvict within the internal key storage structure to TRUE.  
4427     b. Else if bitValue == FALSE  
4428         i. Set ownerEvict within the internal key storage structure to FALSE.  
44296. Return TPM\_SUCCESS



## 443021.2 TPM\_SaveContext

### 4431Start of informative comment:

4432TPM\_SaveContext saves a loaded resource outside the TPM. After successful execution of  
4433the command, the TPM automatically releases the internal memory for sessions but leaves  
4434keys in place.

4435There is no assumption that a saved context blob is stored in a safe, protected area. Since  
4436the context blob can be loaded at any time, do not rely on TPM\_SaveContext to restrict  
4437access to an entity such as a key. If use of the entity should be restricted, means such as  
4438authorization secrets or PCR's should be used.

4439In general, TPM\_SaveContext can save a transport session. However, it cannot save an  
4440exclusive transport session, because any ordinal other than TPM\_ExecuteTransport  
4441terminates the exclusive transport session. This action prevents the exclusive transport  
4442session from being saved and reloaded while intervening commands are hidden from the  
4443transport log.

### 4444End of informative comment.

### 4445Incoming 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

### 4446Outgoing 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

### 4447Description

44481. The caller of the function uses the label field to add additional sequencing, anti-replay or  
4449 other items to the blob. The information does not need to be confidential but needs to be  
4450 part of the blob integrity.

## 4451Actions

44521. Map V1 to TPM\_STANY\_DATA
44532. Validate that handle points to resource that matches resourceType, return  
4454 TPM\_INVALID\_RESOURCE on error
44553. Validate that resourceType is a resource from the following list if not return  
4456 TPM\_INVALID\_RESOURCE
- 4457 a. TPM\_RT\_KEY
- 4458 b. TPM\_RT\_AUTH
- 4459 c. TPM\_RT\_TRANS
- 4460 d. TPM\_RT\_DAA\_TPM
44614. Locate the correct nonce
- 4462 a. If resourceType is TPM\_RT\_KEY
- 4463 i. If TPM\_STCLEAR\_DATA -> contextNonceKey is all zeros
- 4464 (1) Set TPM\_STCLEAR\_DATA -> contextNonceKey to the next value from  
4465 the TPM RNG
- 4466 ii. Map N1 to TPM\_STCLEAR\_DATA -> contextNonceKey
- 4467 iii. If the key has TPM\_KEY\_CONTROL\_OWNER\_EVICT set then return  
4468 TPM\_OWNER\_CONTROL
- 4469 b. Else
- 4470 i. If V1 -> contextNonceSession is all zeros
- 4471 (1) Set V1 -> contextNonceSession to the next value from the TPM RNG
- 4472 ii. Map N1 to V1 -> contextNonceSession
44735. Set K1 to TPM\_PERMANENT\_DATA -> contextKey
44746. Create R1 by putting the sensitive part of the resource pointed to by handle into a  
4475 structure. The structure is a TPM manufacturer option. The TPM MUST ensure that ALL  
4476 sensitive information of the resource is included in R1.
44777. Create C1 a TPM\_CONTEXT\_SENSITIVE structure
- 4478 a. C1 forms the inner encrypted wrapper for the blob. All saved context blobs MUST  
4479 include a TPM\_CONTEXT\_SENSITIVE structure and the TPM\_CONTEXT\_SENSITIVE  
4480 structure MUST be encrypted.
- 4481 b. Set C1 -> contextNonce to N1
- 4482 c. Set C1 -> internalData to R1
44838. Create B1 a TPM\_CONTEXT\_BLOB
- 4484 a. Set B1 -> tag to TPM\_TAG\_CONTEXTBLOB
- 4485 b. Set B1 -> resourceType to resourceType
- 4486 c. Set B1 -> handle to handle
- 4487 d. Set B1 -> integrityDigest to all zeros

1089  
1090  
1091

4488 e. Set B1 -> label to label

4489 f. Set B1 -> additionalData to information determined by the TPM manufacturer. This  
4490 data will help the TPM to reload and reset context. This area MUST NOT hold any data  
4491 that is sensitive (symmetric IV are fine, prime factors of an RSA key are not).

4492 i. For OSAP sessions and for DSAP sessions attached to keys, the hash of the  
4493 entity MUST be included in additionalData

4494 g. Set B1 -> additionalSize to the size of additionalData

4495 h. Set B1 -> sensitiveSize to the size of C1

4496 i. Set B1 -> sensitiveData to C1

44979. If resourceType is TPM\_RT\_KEY

4498 a. Set B1 -> contextCount to 0

449910. Else

4500 a. If V1 -> contextCount >  $2^{32}-2$  then

4501 i. Return with TPM\_TOOMANYCONTEXTS

4502 b. Else

4503 i. Validate that the TPM can still manage the new count value

4504 (1) If the distance between the oldest saved context and the contextCount  
4505 is too large return TPM\_CONTEXT\_GAP

4506 ii. Find contextIndex such that V1 -> contextList[contextIndex] equals 0. If not  
4507 found exit with TPM\_NOCONTEXTSPACE

4508 iii. Increment V1 -> contextCount by 1

4509 iv. Set V1-> contextList[contextIndex] to V1 -> contextCount

4510 v. Set B1 -> contextCount to V1 -> contextCount

4511 c. The TPM MUST invalidate all information regarding the resource except for  
4512 information needed for reloading

451311. Calculate B1 -> integrityDigest the HMAC of B1 using TPM\_PERMANENT\_DATA ->  
4514 tpmProof as the secret

451512. Create E1 by encrypting C1 using K1 as the key

4516 a. Set B1 -> sensitiveSize to the size of E1

4517 b. Set B1 -> sensitiveData to E1

451813. Set contextSize to the size of B1

451914. Return B1 in contextBlob

## 4520**21.3 TPM\_LoadContext**

### 4521**Start of informative comment:**

4522TPM\_LoadContext loads into the TPM a previously saved context. The command returns a  
4523handle.

### 4524**End of informative comment.**

### 4525**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

### 4526**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.

### 4527**Actions**

45281. Map contextBlob to B1, a TPM\_CONTEXT\_BLOB structure

45292. Map V1 to TPM\_STANY\_DATA

45303. Create M1 by decrypting B1 -> sensitiveData using TPM\_PERMANENT\_DATA ->  
4531 contextKey

45324. Create C1 and R1 by splitting M1 into a TPM\_CONTEXT\_SENSITIVE structure and  
4533 internal resource data

45345. Check contextNonce

4535 a. If B1 -> resourceType is NOT TPM\_RT\_KEY

4536 i. If C1 -> contextNonce does not equal V1 -> contextNonceSession return  
4537 TPM\_BADCONTEXT

1099  
1100

4538           ii. Validate that the resource pointed to by the context is loaded (i.e. for OSAP the  
4539           key referenced is loaded and DSAP connected to the key) return  
4540           TPM\_RESOURCEMISSING

4541               (1)    For OSAP sessions and for DSAP sessions attached to keys, the TPM  
4542           MUST validate that the hash of the entity matches the entity held by the TPM

4543               (2)    For OSAP and DSAP sessions referring to a key, verify that entityHandle  
4544           identifies the key linked to this OSAP/DSAP session, if not return  
4545           TPM\_BAD\_HANDLE.

4546    b. Else

4547           i. If C1 -> internalData -> parentPCRStatus is FALSE and C1 -> internalData ->  
4548           isVolatile is FALSE

4549               (1)    Ignore C1 -> contextNonce

4550           ii. else

4551               (1)    If C1 -> contextNonce does not equal TPM\_STCLEAR\_DATA ->  
4552           contextNonceKey return TPM\_BADCONTEXT

45536. Validate the structure

4554    a. Set H1 to B1 -> integrityDigest

4555    b. Set B1 -> integrityDigest to all zeros

4556    c. Copy M1 to B1 -> sensitiveData

4557    d. Create H2 the HMAC of B1 using TPM\_PERMANENT\_DATA -> tpmProof as the HMAC  
4558    key

4559    e. If H2 does not equal H1 return TPM\_BADCONTEXT

45607. If keepHandle is TRUE

4561    a. Set handle to B1 -> handle

4562    b. If the TPM is unable to restore the handle the TPM MUST return TPM\_BAD\_HANDLE

45638. Else

4564    a. The TPM SHOULD attempt to restore the handle but if not possible it MAY set the  
4565    handle to any valid for B1 -> resourceType

45669. If B1 -> resourceType is NOT TPM\_RT\_KEY

4567    a. Find contextIndex such that V1 -> contextList[contextIndex] equals B1 ->  
4568    TPM\_CONTEXT\_BLOB -> contextCount

4569    b. If not found then return TPM\_BADCONTEXT

4570    c. Set V1 -> contextList[contextIndex] to 0

457110. Process B1 to return the resource back into TPM use

## 4572**22. Eviction**

### 4573**Start of informative comment:**

4574The TPM has numerous resources held inside of the TPM that may need eviction. The need  
4575for eviction occurs when the number of resources in use by the TPM exceed the available  
4576space. For resources that are hard to reload (i.e. keys tied to PCR values) the outside entity  
4577should first perform a context save before evicting items.

4578In version 1.1 there were separate commands to evict separate resource types. This new  
4579command set uses the resource types defined for context saving and creates a generic  
4580command that will evict all resource types.

### 4581**End of informative comment.**

4582The TPM MUST NOT flush the EK or SRK using this command.

4583Version 1.2 deprecates the following commands:

- 4584• TPM\_Terminate\_Handle
- 4585• TPM\_EvictKey
- 4586• TPM\_Reset

## 4587**22.1 TPM\_FlushSpecific**

### 4588**Start of informative comment:**

4589TPM\_FlushSpecific flushes from the TPM a specific handle.

### 4590**End of informative comment.**

### 4591**Incoming Parameters and Sizes**

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

### 4592**Outgoing Parameters and Sizes**

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

### 4593**Description**

4594TPM\_FlushSpecific releases the resources associated with the given handle.

### 4595**Actions**

45961. If resourceType is TPM\_RT\_CONTEXT

4597 a. The handle for a context is not a handle but the "context count" value. The TPM uses  
4598 the "context count" value to locate the proper contextList entry and sets R1 to the  
4599 contextList entry

46002. Else if resourceType is TPM\_RT\_KEY

4601 a. Set R1 to the key pointed to by handle  
4602 b. If R1 -> ownerEvict is TRUE return TPM\_KEY\_OWNER\_CONTROL

46033. Else if resourceType is TPM\_RT\_AUTH

4604 a. Set R1 to the authorization session pointed to by handle

46054. Else if resourceType is TPM\_RT\_TRANS

4606 a. Set R1 to the transport session pointed to by handle

46075. Else if resourceType is TPM\_RT\_DAA\_TPM

- 4608    a. Set R1 to the DAA session pointed to by handle
- 46096. Else return TPM\_INVALID\_RESOURCE
- 46107. Validate that R1 determined by resourceType and handle points to a valid allocated
- 4611    resource. Return TPM\_BAD\_PARAMETER on error.
- 46128. Invalidate R1 and all internal resources allocated to R1
- 4613    a. Resources include authorization sessions



## 4614**23. Timing Ticks**

### 4615**Start of informative comment:**

4616The TPM timing ticks are always available for use. The association of timing ticks to actual  
4617time is a protocol that occurs outside of the TPM. See the design document for details.

4618The setting of the clock type variable is a one time operation that allows the TPM to be  
4619configured to the type of platform that is installed on.

4620The ability for the TPM to continue to increment the timer ticks across power cycles of the  
4621platform is a TPM and platform manufacturer decision.

### 4622**End of informative comment.**

## 4623**23.1 TPM\_GetTicks**

### 4624**Start of informative comment:**

4625This command returns the current tick count of the TPM.

### 4626**End of informative comment.**

### 4627**Incoming Parameters and Sizes**

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

### 4628**Outgoing Parameters and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Ordinal: TPM_ORD_GetTicks
4	32	3S	32	TPM_CURRENT_TICKS	currentTime	The current time held in the TPM

### 4629**Description**

4630This command returns the current time held in the TPM. It is the responsibility of the  
4631external system to maintain any relation between this time and a UTC value or local real  
4632time value.

### 4633**Actions**

46341. Set T1 to the internal TPM\_CURRENT\_TICKS structure

46352. Return T1 as currentTime.

## 4636**23.2 TPM\_TickStampBlob**

### 4637**Start of informative comment:**

4638This command applies a time stamp to the passed blob. The TPM makes no representation  
4639regarding the blob merely that the blob was present at the TPM at the time indicated.

### 4640**End of informative comment.**

### 4641**Incoming Parameters and Sizes**

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

## 4642Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Ordinal, fixed value of TPM_ORD_TickStampBlob
4	32	3S	32	TPM_CURRENT_TICKS	currentTicks	The current time according to the TPM
5	4	4S	4	UINT32	sigSize	The length of the returned digital signature
6	<>	5S	<>	BYTE[]	sig	The resulting digital signature.
7	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
9	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: key.usageAuth

## 4643Description

4644The function performs a digital signature on the hash of digestToStamp and the current tick  
4645count.

4646It is the responsibility of the external system to maintain any relation between tick count  
4647and a UTC value or local real time value.

## 4648Actions

46491. The TPM validates the AuthData to use the key pointed to by keyHandle.
46502. Validate that keyHandle -> keyUsage is TPM\_KEY\_SIGNING, TPM\_KEY\_IDENTITY or  
4651 TPM\_KEY\_LEGACY, if not return the error code TPM\_INVALID\_KEYUSAGE.
46523. Validate that keyHandle -> sigScheme is TPM\_SS\_RSASSAPKCS1v15\_SHA1 or  
4653 TPM\_SS\_RSASSAPKCS1v15\_INFO, if not return TPM\_INAPPROPRIATE\_SIG.
46544. If TPM\_STCLEAR\_DATA -> currentTicks is not properly initialized  
4655 a. Initialize the TPM\_STCLEAR\_DATA -> currentTicks
46565. Create T1, a TPM\_CURRENT\_TICKS structure.
46576. Create H1 a TPM\_SIGN\_INFO structure and set the structure defaults  
4658 a. Set H1 -> fixed to "TSTP"  
4659 b. Set H1 -> replay to antiReplay  
4660 c. Create H2 the concatenation of digestToStamp || T1  
4661 d. Set H1 -> dataLen to the length of H2  
4662 e. Set H1 -> data to H2

46637. The TPM computes the signature, sig, using the key referenced by keyHandle, using  
4664 SHA-1 of H1 as the information to be signed
46658. The TPM returns T1 as currentTicks parameter

4666**24. Transport Sessions**  
4667

4668See Part 1 for rationale and security issues.

4669**24.1 TPM\_EstablishTransport**

4670**Start of informative comment:**

4671This establishes the transport session. Depending on the attributes specified for the session  
4672this may establish shared secrets, encryption keys, and session logs. The session will be in  
4673use for by the TPM\_ExecuteTransport command.

4674The only restriction on what can happen inside of a transport session is that there is no  
4675“nesting” of sessions. It is permissible to perform operations that delete internal state and  
4676make the TPM inoperable.

4677**End of informative comment.**

4678**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

4679

## 4680Outgoing 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

## 4681Description

4682This command establishes the transport sessions shared secret. The encryption of the  
4683shared secret uses the public key of the key loaded in encKey.

## 4684Actions

46851. If encHandle is TPM\_KH\_TRANSPORT then

4686 a. If tag is NOT TPM\_TAG\_RQU\_COMMAND return TPM\_BADTAG

4687 b. If transPublic -> transAttributes specifies TPM\_TRANSPORT\_ENCRYPT return  
4688 TPM\_BAD\_SCHEME

4689 c. If secretSize is not 20 return TPM\_BAD\_PARAM\_SIZE

4690 d. Set A1 to secret

46912. Else

4692 a. encHandle -> keyUsage MUST be TPM\_KEY\_STORAGE or TPM\_KEY\_LEGACY return  
4693 TPM\_INVALID\_KEYUSAGE on error

4694 b. If encHandle -> authDataUsage does not equal TPM\_AUTH\_NEVER and tag is NOT  
4695 TPM\_TAG\_RQU\_AUTH1\_COMMAND return TPM\_AUTHFAIL

4696 c. Using encHandle -> usageAuth validate the AuthData to use the key and the  
4697 parameters to the command

4698 d. Create K1 a TPM\_TRANSPORT\_AUTH structure by decrypting secret using the key  
4699 pointed to by encHandle

4700 e. Validate K1 for tag

4701 f. Set A1 to K1 -> authData

1144

1145

47023. If transPublic -> transAttributes has TPM\_TRANSPORT\_ENCRYPT
- 4703 a. If TPM\_PERMANENT\_FLAGS -> FIPS is true and transPublic -> algId is equal to
- 4704 TPM\_ALG\_MGF1 return TPM\_INAPPROPRIATE\_ENC
- 4705 b. Check if the transPublic -> algId is supported, if not return
- 4706 TPM\_BAD\_KEY\_PROPERTY
- 4707 c. If transPublic -> algId is TPM\_ALG\_AESXXX, check that transPublic -> encScheme is
- 4708 supported, if not return TPM\_INAPPROPRIATE\_ENC
- 4709 d. Perform any initializations necessary for the algorithm
47104. Generate transNonceEven from the TPM RNG
47115. Create T1 a TPM\_TRANSPORT\_INTERNAL structure
- 4712 a. Ensure that the TPM has sufficient internal space to allocate the transport session,
- 4713 return TPM\_RESOURCES on error
- 4714 b. Assign a T1 -> transHandle value. This value is assigned by the TPM
- 4715 c. Set T1 -> transDigest to all zeros
- 4716 d. Set T1 -> transPublic to transPublic
- 4717 e. Set T1-> transNonceEven to transNonceEven
- 4718 f. Set T1 -> authData to A1
47196. If TPM\_STANY\_DATA -> currentTicks is not properly initialized
- 4720 a. Initialize the TPM\_STANY\_DATA -> currentTicks
47217. Set currentTicks to TPM\_STANY\_DATA -> currentTicks
47228. If T1 -> transPublic -> transAttributes has TPM\_TRANSPORT\_LOG set then
- 4723 a. Create L1 a TPM\_TRANSPORT\_LOG\_IN structure
- 4724 i. Set L1 -> parameters to SHA-1 (ordinal || transPublic || secretSize || secret)
- 4725 ii. Set L1 -> pubKeyHash to all zeros
- 4726 iii. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || L1)
- 4727 b. Create L2 a TPM\_TRANSPORT\_LOG\_OUT structure
- 4728 i. Set L2 -> parameters to SHA-1 (returnCode || ordinal || locality ||
- 4729 currentTicks || transNonceEven)
- 4730 ii. Set L2 -> locality to the locality of this command
- 4731 iii. Set L2 -> currentTicks to currentTicks, this MUST be the same value that is
- 4732 returned in the currentTicks parameter
- 4733 iv. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || L2)
47349. If T1 -> transPublic -> transAttributes has TPM\_TRANSPORT\_EXCLUSIVE then set
- 4735 TPM\_STANY\_FLAGS -> transportExclusive to TRUE
- 4736 a. Execution of any command other than TPM\_ExecuteTransport or
- 4737 TPM\_ReleaseTransportSigned targeting this transport session will cause the abnormal
- 4738 invalidation of this transport session transHandle

4739     b. The TPM gives no indication, other than invalidation of transHandle, that the session  
4740     is terminated

474110.Return T1 -> transHandle as transHandle



## 4742**24.2 TPM\_ExecuteTransport**

### 4743**Start of informative comment:**

4744Delivers a wrapped TPM command to the TPM where the TPM unwraps the command and  
4745then executes the command.

4746TPM\_ExecuteTransport uses the same rolling nonce paradigm as other authorized TPM  
4747commands. The even nonces start in TPM\_EstablishTransport and change on each  
4748invocation of TPM\_ExecuteTransport.

4749The only restriction on what can happen inside of a transport session is that there is no  
4750“nesting” of sessions. It is permissible to perform operations that delete internal state and  
4751make the TPM inoperable.

4752Because, in general, key handles are not logged, a digest of the corresponding public key is  
4753logged. In cases where the key handle is logged (e.g. TPM\_OwnerReadInternalPub), the  
4754public key is also logged.

4755The wrapped command is audited twice – once according to the actions of  
4756TPM\_ExecuteTransport and once within the wrapped command itself according to the  
4757special rules for auditing a command wrapped in an encrypted transport session.

4758The method of incrementing the symmetric key counter value is different from that used by  
4759some standard crypto libraries (e.g. openssl, Java JCE) that increment the entire counter  
4760value. TPM users should be aware of this to avoid errors when the counter wraps.

### 4761**End of informative comment.**

### 4762**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

4763

#### 4764Outgoing 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

#### 4765Description

47661. This command executes a TPM command using the transport session.
47672. Prior to execution of the wrapped command (action 11 below) failure of the transport session MUST have no effect on the resources referenced by the wrapped command. The exception is when the TPM goes into failure mode and return FAILED\_SELFTEST for all subsequent commands.
47713. After execution of the wrapped command, failure of the transport session MAY NOT affect wrapped command resources. That is, the TPM is not required to clean up the effects of the wrapped command. Sessions and keys MAY remain loaded. It is understood that the transport session will be returning an error code and not reporting any session nonces. Therefore, wrapped sessions are no longer useful to the caller. It is the responsibility of the caller to clean up the result of the wrapped command.
47774. Execution of the wrapped command (action 11) SHOULD have no effect on the transport session.
- 4779 a. The wrapped command SHALL use no resources of the transport session, this includes authorization sessions
- 4781 b. If the wrapped command execution returns an error (action 11 below) then the sessions for TPM\_ExecuteTransport still operate properly.
- 4783 c. The exception to this is when the wrapped command causes the TPM to go into failure mode and return TPM\_FAILSELFTEST for all subsequent commands
47855. Field layout
- 4786 a. Notation
- 4787i. et indicates the outer TPM\_ExecuteTransport command and response

1162

1163

4788ii. w indicates the inner command and response that is wrapped by the  
4789 TPM\_ExecuteTransport.

4790iii. (o) indicates optional parameters that may or may not be present in the wrapped  
4791 command.

4792 b. Command representation

4793 c. \*\*\*\*\*

4794 d. TAGet | LENet | ORDet | wrappedCmdSize | wrappedCmd | AUTHet

4795 e. \*\*\*\*\*

4796 f. wrappedCmd looks like the following

4797 g. \*\*\*\*\*

4798 h. TAGw | LENw | ORDw | HANDLESw(o) | DATAw | AUTH1w (o) | AUTH2w (o)

4799 i. \*\*\*\*\*

4800 j. | LEN1 |

4801 k. | E1 | (encrypted)

4802 l. | C1 | (decrypted)

4803 m. Response representation

4804 n. \*\*\*\*\*

4805 o. TAGet | LENet | RCet | ... | wrappedRspSize | wrappedRsp | AUTHet

4806 p. \*\*\*\*\*

4807 q. wrappedRsp looks like the following

4808 r. \*\*\*\*\*

4809 s. TAGw | LENw | RCw | HANDLESw(o) | DATAw | AUTH1w (o) | AUTH2w (o)

4810 t. \*\*\*\*\*

4811 u. | LEN2 |

4812 v. | ←----- C2 -----→ |

4813 w. | S2 | (decrypted)

4814 x. | E2 | (encrypted)

4815 y. The only command and response parameter that is possibly encrypted is DATAw.

48166. Additional DATAw comments

4817 a. For TPM\_FlushSpecific and TPM\_SaveContext

4818 i. The DATAw part of these commands does not include the handle.

4819 (1) It is understood that encrypting the resourceType prevents a determination of  
4820 the handle type.

4821 ii. If the resourceType is TPM\_RT\_KEY, then the public key MUST be logged.

4822 b. For TPM\_DAA\_Join and TPM\_DAA\_Sign

- 4823 i. The DATAw part of these commands does not include the input handle. The  
4824 output handle from stage 0 is included in DATAw.
- 4825 c. For TPM\_LoadKey2
- 4826 i. The outgoing handle is not part of the outgoing DATAw and is not encrypted or  
4827 logged by the outgoing transport.
- 4828 d. For TPM\_LoadKey
- 4829 i. The outgoing handle is part of the outgoing DATAw and is encrypted and  
4830 logged.
- 4831 e. For TPM\_LoadContext
- 4832 i. The outgoing handle is not part of the outgoing DATAw and is not encrypted or  
4833 logged by the outgoing transport.
- 4834 (1) It is understood that encrypting the contextBlob prevents a determination of  
4835 the handle type.
- 4836 f. For TPM\_OIAP and TPM\_OSAP, no input or output parameters are encrypted  
4837 or logged.
- 4838 i. For TPM\_OSAP, the public key MUST NOT be logged. During a logged  
4839 transport session, when a wrapped command uses the key, the public key  
4840 referenced by the key handle will be logged in the transport. Thus, the audit trail  
4841 is established for any key usage at that time.
- 4842 g. For TPM\_DSAP
- 4843 i. For input, only entityValue is encrypted and logged.
- 4844 ii. For output no parameters are encrypted or logged.
- 4845 7. TPM\_ExecuteTransport returns an implementation defined result when the wrapped  
4846 command would cause termination of the transport session. Implementation defined  
4847 possibilities include but are not limited to: the wrapped command may execute,  
4848 completely, partially, or not at all, the transport session may or may not be terminated,  
4849 continueTransSession may not be processed or returned correctly, and an error may or  
4850 may not be returned. The wrapped commands include:
- 4851 a. TPM\_FlushSpecific, TPM\_SaveContext targeting the transport session
- 4852 b. TPM\_OwnerClear, TPM\_ForceClear, TPM\_RevokeTrust

### 4853Actions

- 4854 1. Using transHandle locate the TPM\_TRANSPORT\_INTERNAL structure T1
- 4855 2. Parse wrappedCmd
- 4856 a. Set TAGw, LENw, and ORDw to the parameters from wrappedCmd
- 4857 b. Set E1 to DATAw
- 4858 i. This pointer is ordinal dependent and requires the execute transport  
4859 command to parse wrappedCmd
- 4860 c. Set LEN1 to the length of DATAw

- 1171  
1172
- 4861 i. DATAw always ends at the start of AUTH1w if AUTH1w is present
48623. If LEN1 is less than 0, or if ORDw is unknown, unimplemented, or cannot be determined
- 4863 a. Return TPM\_BAD\_PARAMETER
48644. If T1 -> transPublic -> transAttributes has TPM\_TRANSPORT\_ENCRYPT set then
- 4865 a. If T1 -> transPublic -> algId is TPM\_ALG\_MGF1
- 4866 i. Using the MGF1 function, create string G1 of length LEN1. The inputs to the
- 4867 MGF1 are transLastNonceEven, transNonceOdd, "in", and T1 -> authData. These
- 4868 four values concatenated together form the Z value that is the seed for the MGF1.
- 4869 ii. Create C1 by performing an XOR of G1 and wrappedCmd starting at E1.
- 4870 b. If the encryption algorithm requires an IV or CTR, calculate the IV or CTR value
- 4871 i. Using the MGF1 function, create string IV1 or CTR1 with a length set by the
- 4872 block size of the encryption algorithm. The inputs to the MGF1 are
- 4873 transLastNonceEven, transNonceOdd, and "in". These three values concatenated
- 4874 together form the Z value that is the seed for the MGF1. Note that any
- 4875 terminating characters within the string "in" are ignored, so a total of 42 bytes are
- 4876 hashed.
- 4877 ii. The symmetric key is taken from the first bytes of T1 -> authData.
- 4878 iii. Decrypt DATAw and replace the DATAw area of E1 creating C1
- 4879 c. TPM\_OSAP, TPM\_OIAP have no parameters encrypted
- 4880 d. TPM\_DSAP has special rules for parameter encryption
48815. Else
- 4882 a. Set C1 to the DATAw area E1 of wrappedCmd
48836. Create H1 the SHA-1 of (ORDw || C1).
- 4884 a. C1 MUST point at the decrypted DATAw area of E1
- 4885 b. The TPM MAY use this calculation for both execute transport authorization,
- 4886 authorization of the wrapped command and transport log creation
48877. Validate the incoming transport session authorization
- 4888 a. Set inParamDigest to SHA-1 (ORDet || wrappedCmdSize || H1)
- 4889 b. Calculate the HMAC of (inParamDigest || transLastNonceEven || transNonceOdd ||
- 4890 continueTransSession) using T1 -> authData as the HMAC key
- 4891 c. Validate transAuth, on errors return TPM\_AUTHFAIL
48928. If TPM\_ExecuteTransport requires auditing
- 4893 a. Create TPM\_AUDIT\_EVENT\_IN using H1 as the input parameter digest and update
- 4894 auditDigest
- 4895 b. On any error return TPM\_AUDITFAIL\_UNSUCCESSFUL
48969. If ORDw is from the list of following commands return TPM\_NO\_WRAP\_TRANSPORT
- 4897 a. TPM\_EstablishTransport

- 4898    b. TPM\_ExecuteTransport  
4899    c. TPM\_ReleaseTransportSigned
- 490010.If T1 -> transPublic -> transAttributes has TPM\_TRANSPORT\_LOG set then
- 4901    a. Create L2 a TPM\_TRANSPORT\_LOG\_IN structure  
4902    b. Set L2 -> parameters to H1  
4903    c. If ORDw is a command with no key handles  
4904        i. Set L2 -> pubKeyHash to all zeros  
4905    d. If ORDw is a command with one key handle  
4906        i. Create K2 the hash of the TPM\_STORE\_PUBKEY structure of the key pointed  
4907           to by the key handle.  
4908        ii. Set L2 -> pubKeyHash to SHA-1 (K2)  
4909    e. If ORDw is a command with two key handles  
4910        i. Create K2 the hash of the TPM\_STORE\_PUBKEY structure of the key pointed  
4911           to by the first key handle.  
4912        ii. Create K3 the hash of the TPM\_STORE\_PUBKEY structure of the key pointed  
4913           to by the second key handle.  
4914        iii. Set L2 -> pubKeyHash to SHA-1 (K2 || K3)  
4915    f. Set T1 -> transDigest to the SHA-1 (T1 -> transDigest || L2)  
4916    g. If ORDw is a command with key handles, and the key is not loaded, return  
4917        TPM\_INVALID\_KEYHANDLE.
- 491811.Send the wrapped command to the normal TPM command parser, the output is C2 and  
4919    the return code is RCw
- 4920    a. If ORDw is a command that is audited then the TPM MUST perform the input and  
4921       output audit of the command as part of this action.  
4922    b. The TPM MAY use H1 as the data value in the authorization and audit calculations  
4923       during the execution of C1
- 492412.Set CT1 to TPM\_STANY\_DATA -> currentTicks -> currentTicks and return CT1 in the  
4925    currentTicks output parameter
- 492613.Calculate S2 the pointer to the DATAw area of C2  
4927    a. Calculate LEN2 the length of S2 according to the same rules that calculated LEN1
- 492814.Create H2 the SHA-1 of (RCw || ORDw || S2)  
4929    a. The TPM MAY use this calculation for execute transport authorization and transport  
4930       log out creation
- 493115.Calculate the outgoing transport session authorization  
4932    a. Create the new transNonceEven for the output of the command  
4933    b. Set outParamDigest to SHA-1 (RCet || ORDet || TPM\_STANY\_DATA -> currentTicks  
4934       -> currentTicks || locality || wrappedRspSize || H2)

1180

1181

4935 c. Calculate transAuth, the HMAC of (outParamDigest || transNonceEven ||  
 4936 transNonceOdd || continueTransSession) using T1 -> authData as the HMAC key

4937 16.If T1 -> transPublic -> transAttributes has TPM\_TRANSPORT\_LOG set then

4938 a. Create L3 a TPM\_TRANSPORT\_LOG\_OUT structure

4939 b. Set L3 -> parameters to H2

4940 c. Set L3 -> currentTicks to TPM\_STANY\_DATA -> currentTicks

4941 d. Set L3 -> locality to TPM\_STANY\_DATA -> localityModifier

4942 e. Set T1 -> transDigest to the SHA-1 (T1 -> transDigest || L3)

4943 17.If T1 -> transPublic -> transAttributes has TPM\_TRANSPORT\_ENCRYPT set then

4944 a. If T1 -> transPublic -> algId is TPM\_ALG\_MGF1

4945 i. Using the MGF1 function, create string G2 of length LEN2. The inputs to the  
 4946 MGF1 are transNonceEven, transNonceOdd, “out”, and T1 -> authData. These  
 4947 four values concatenated together form the Z value that is the seed for the MGF1.

4948 ii. Create E2 by performing an XOR of G2 and C2 starting at S2.

4949 b. Else

4950 i. Create IV2 or CTR2 using the same algorithm as IV1 or CTR1 with the input  
 4951 values transNonceEven, transNonceOdd, and “out”. Note that any terminating  
 4952 characters within the string “out” are ignored, so a total of 43 bytes are hashed.

4953 ii. The symmetric key is taken from the first bytes of T1 -> authData

4954 iii. Create E2 by encrypting C2 starting at S2

4955 18.Else

4956 a. Set E2 to the DATAw area S2 of wrappedRsp

4957 19.If continueTransSession is FALSE

4958 a. Invalidate all session data related to transHandle

4959 20.If TPM\_ExecuteTransport requires auditing

4960 a. Create TPM\_AUDIT\_EVENT\_OUT using H2 for the parameters and update the  
 4961 auditDigest

4962 b. On any errors return TPM\_AUDITFAIL\_SUCCESSFUL or  
 4963 TPM\_AUDITFAIL\_UNSUCCESSFUL depending on RCw

4964 21.Return C2 but with S2 replaced by E2 in the wrappedRsp parameter

## 4965**24.3 TPM\_ReleaseTransportSigned**

### 4966**Start of informative comment:**

4967This command completes the transport session. If logging for this session is turned on, then  
4968this command returns a digital signature of the hash of all operations performed during the  
4969session.

4970This command serves no purpose if logging is turned off, and results in an error if  
4971attempted.

4972This command uses two authorization sessions, the key that will sign the log and the  
4973authorization from the session. Having the session authorization proves that the requestor  
4974that is signing the log is the owner of the session. If this restriction is not put in then an  
4975attacker can close the log and sign using their own key.

4976The hash of the session log includes the information associated with the input phase of  
4977execution of the TPM\_ReleaseTransportSigned command. It cannot include the output  
4978phase information.

### 4979**End of informative comment.**

### 4980**Incoming Parameters and Sizes**

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



## 4981Outgoing Parameters and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH2_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ReleaseTransportSigned
4	4	3S	4	TPM_MODIFIER_INDICATOR	locality	The locality that called this command
5	32	4S	32	TPM_CURRENT_TICKS	currentTicks	The current ticks when the command executed
6	4	5S	4	UINT32	signSize	The size of the signature area
7	<>	6S	<>	BYTE[]	signature	The signature of the digest
8	20	2H1	20	TPM_NONCE	authNonceEven	Even nonce newly generated by TPM
		3H1	20	TPM_NONCE	authNonceOdd	Nonce generated by caller
9	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the session
10	20			TPM_AUTHDATA	keyAuth	HMAC: key -> usageAuth
11	20	2H2	20	TPM_NONCE	transNonceEven	Even nonce newly generated by TPM
		3H2	20	TPM_NONCE	transNonceOdd	Nonce generated by caller
12	1	4H2	1	BOOL	continueTransSession	The continue use flag for the session
13	20			TPM_AUTHDATA	transAuth	HMAC: transHandle -> authData

## 4982Description

4983This command releases a transport session and signs the transport log

## 4984Actions

49851. Using transHandle locate the TPM\_TRANSPORT\_INTERNAL structure T1
49862. Validate that keyHandle -> sigScheme is TPM\_SS\_RSASSAPKCS1v15\_SHA1 or  
4987 TPM\_SS\_RSASSAPKCS1v15\_INFO, if not return TPM\_INAPPROPRIATE\_SIG.
49883. Validate that keyHandle -> keyUsage is TPM\_KEY\_SIGNING, if not return  
4989 TPM\_INVALID\_KEYUSAGE
49904. Using key -> authData validate the command and parameters, on error return  
4991 TPM\_AUTHFAIL
49925. Using transHandle -> authData validate the command and parameters, on error return  
4993 TPM\_AUTH2FAIL
49946. If T1 -> transAttributes has TPM\_TRANSPORT\_LOG set then
  - 4995 a. Create A1 a TPM\_TRANSPORT\_LOG\_OUT structure
  - 4996 b. Set A1 -> parameters to the SHA-1 (ordinal || antiReplay)
  - 4997 c. Set A1 -> currentTicks to TPM\_STANY\_DATA -> currentTicks
  - 4998 d. Set A1 -> locality to the locality modifier for this command

- 4999 e. Set T1 -> transDigest to SHA-1 (T1 -> transDigest || A1)
- 50007. Else
- 5001 a. Return TPM\_BAD\_MODE
- 50028. Create H1 a TPM\_SIGN\_INFO structure and set the structure defaults
- 5003 a. Set H1 -> fixed to “TRAN”
- 5004 b. Set H1 -> replay to antiReplay
- 5005 c. Set H1 -> data to T1 -> transDigest
- 5006 d. Sign SHA-1 hash of H1 using the key pointed to by keyHandle
- 50079. Invalidate all session data related to T1
- 500810. Set continueTransSession to FALSE
- 500911. Return TPM\_SUCCESS

## 501025. Monotonic Counter

### 501125.1 TPM\_CreateCounter

#### 5012Start of informative comment:

5013This command creates the counter but does not select the counter. Counter creation  
5014assigns an AuthData value to the counter and sets the counters original start value. The  
5015original start value is the current internal base value plus one. Setting the new counter to  
5016the internal base avoids attacks on the system that are attempting to use old counter  
5017values.

#### 5018End of informative comment.

#### 5019Incoming 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.

#### 5020Outgoing 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.

#### 5021Description

5022This command creates a new monotonic counter. The TPM MUST support a minimum of 4  
5023concurrent counters.

#### 5024**Actions**

5025The TPM SHALL do the following:

50261. Using the authHandle field, validate the owner's AuthData to execute the command and  
5027 all of the incoming parameters. The authorization session MUST be OSAP or DSAP
50282. Ignore continueAuthSession on input and set continueAuthSession to FALSE on output
50293. Create a1 by decrypting encAuth according to the ADIP indicated by authHandle.
50304. Validate that there is sufficient internal space in the TPM to create a new counter. If  
5031 there is insufficient space, the command returns an error.
- 5032 a. The TPM MUST provide storage for a1, TPM\_COUNTER\_VALUE, countID, and any  
5033 other internal data the TPM needs to associate with the counter
50345. Increment the max counter value
50356. Set the counter to the max counter value
50367. Set the counter label to label
50378. Create a countID

## 503825.2 TPM\_IncrementCounter

### 5039Start of informative comment:

5040This authorized command increments the indicated counter by one. Once a counter has  
5041been incremented then all subsequent increments must be for the same handle until a  
5042successful TPM\_Startup(ST\_CLEAR) is executed.

5043The order for checking validation of the command parameters when no counter is active,  
5044keeps an attacker from creating a denial-of-service attack.

### 5045End of informative comment.

### 5046Incoming 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

### 5047Outgoing 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

### 5048Description

5049This function increments the counter by 1.

5050The TPM MAY implement increment throttling to avoid burn problems

## 5051**Actions**

50521. If TPM\_STCLEAR\_DATA -> countID is 0

5053     a. Validate that countID is a valid counter, return TPM\_BAD\_COUNTER on mismatch

5054     b. Validate the command parameters using counterAuth

5055     c. Set TPM\_STCLEAR\_DATA -> countID to countID

50562. else

5057     a. If TPM\_STCLEAR\_DATA -> countID does not equal countID

5058         i. Return TPM\_BAD\_COUNTER

5059     b. Validate the command parameters using counterAuth

50603. Increments the counter by 1

50614. Return new count value in count

## 5062**25.3 TPM\_ReadCounter**

### 5063**Start of informative comment:**

5064Reading the counter provides the caller with the current number in the sequence.

### 5065**End of informative comment.**

### 5066**Incoming Parameters and Sizes**

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

### 5067**Outgoing Parameters and Sizes**

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

### 5068**Description**

5069This returns the current value for the counter indicated. The counter MAY be any valid  
5070counter.

### 5071**Actions**

50721. Validate that countID points to a valid counter. Return TPM\_BAD\_COUNTER on error.

50732. Return count

## 5074**25.4 TPM\_ReleaseCounter**

### 5075**Start of informative comment:**

5076This command releases a counter such that no reads or increments of the indicated counter  
5077will succeed.

### 5078**End of informative comment.**

### 5079**Incoming Parameters and Sizes**

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

### 5080**Outgoing Parameters and Sizes**

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

### 5081**Actions**

5082The TPM uses countID to locate a valid counter.

50831. Authenticate the command and the parameters using the AuthData pointed to by  
5084 countID. Return TPM\_AUTHFAIL on error

50852. The TPM invalidates all internal information regarding the counter. This includes  
5086 releasing countID such that any subsequent attempts to use countID will fail.

50873. The TPM invalidates sessions



1225

1226

- 5088     a. MUST invalidate all OSAP sessions associated with the counter
- 5089     b. MAY invalidate any other session
- 5090 4. If TPM\_STCLEAR\_DATA -> countID equals countID,
- 5091     a. Set TPM\_STCLEAR\_DATA -> countID to an illegal value (not the zero value)

## 509225.5 TPM\_ReleaseCounterOwner

### 5093Start of informative comment:

5094This command releases a counter such that no reads or increments of the indicated counter  
5095will succeed.

### 5096End of informative comment.

### 5097Incoming 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

### 5098Outgoing 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

### 5099Description

5100This invalidates all information regarding a counter.

### 5101Actions

51021. Validate that ownerAuth properly authorizes the command and parameters

51032. The TPM uses countID to locate a valid counter. Return TPM\_BAD\_COUNTER if not  
5104 found.

1234

1235

51053. The TPM invalidates all internal information regarding the counter. This includes  
5106 releasing countID such that any subsequent attempts to use countID will fail.

51074. The TPM invalidates sessions

5108 a. MUST invalidate all OSAP sessions associated with the counter

5109 b. MAY invalidate any other session

51105. If TPM\_STCLEAR\_DATA -> countID equals countID,

5111 a. Set TPM\_STCLEAR\_DATA -> countID to an illegal value (not the zero value)

## 511226. DAA commands

### 511326.1 TPM\_DAA\_Join

#### 5114Start of informative comment:

5115TPM\_DAA\_Join is the process that establishes the DAA parameters in the TPM for a specific  
5116DAA issuing authority.

5117outputSize and outputData are always included in the outParamDigest. This includes stage  
51180, where the outputData contains the DAA session handle.

#### 5119End of informative comment.

### 5120Incoming 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.

5121Outgoing Operands and Sizes

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

## 5122Description

5123This table summaries the input, output and saved data that is associated with each stage of  
5124processing.

Stage	Input Data0	Input Data1	Operation	Output Data	Scratchpad
0	DAA_count (used as # repetitions of stage 1)	NULL	initialise	Session Handle	NULL
1	n0	signatureValue	rekeying	NULL	n0
2	DAA_issuerSettings	signatureValue	issuer settings	NULL	NULL
3	DAA_count	NULL	DAA_join_uo, DAA_join_u1	NULL	NULL
4	DAA_generic_R0	DAA_generic_n	$P1 = R0^{f0} \bmod n$	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	$r = r0 + (2^{\text{power}0}) \cdot r1 \bmod q,$ $E1 = w^r \bmod \text{gamma}$	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	$s2 = r2 + c^{u0}$ $\bmod 2^{\text{power}1}$	s2	NULL
20	NULL	NULL	$s12 = r2 + c^{u0}$ $\gg \text{power}1$	c	s12
21	NULL	NULL	$s3 = r3 + c^{u1} + s12$	s3	NULL
22	u2	NULL	$v0 = u2 + u0 \bmod 2^{\text{power}1}$ $v10 = u2 + u0 \gg \text{power}1$	enc(v0)	v10
23	u3	NULL	$V1 = u3 + u1 + v10$	enc(v1)	NULL
24	NULL	NULL	enc(DAA_tpmSpecific)	enc(DAA_tpmSpecific)	NULL

5125

## 5126 Actions

5127 A Trusted Platform Module that receives a valid TPM\_DAA\_Join command SHALL:

5128 1. Use ownerAuth to verify that the Owner authorized all TPM\_DAA\_Join input parameters.

5129 2. Any error return results in the TPM invalidating all resources associated with the join

5130 3. Constant values of 0 or 1 are 1 byte integers, stages affected are

5131 a. 4(j), 5(j), 14(f), 17(e)

5132 4. Representation of the strings “r0” to “r4” are 2-byte ASCII encodings, stages affected are

5133 a. 9(i), 10(h), 11(h), 12(h), 15(f), 15(g), 17(d), 18(d), 19(d), 20(d), 21(d)

### 5134 Start of informative comment:

5135 5. Variable DAA\_Count

5136 a. In stage 0, DAA\_Count denotes the length of the RSA key chain, which certifies the  
5137 main DAA public key and which will be loaded in stage 1. It also denotes the number of  
5138 times stage 1 is executed.

5139 b. In stage 3 the variable DAA\_count denotes the actual DAA counter. It allows a DAA  
5140 issuer to keep track of the number of times it has issued 'different' DAA credentials to  
5141 the same platform. (The counter does not need to be equal to the actual number.)

### 5142 End of informative comment.

## 5143 Stages

5144 0. If stage==0

5145 c. Determine that sufficient resources are available to perform a TPM\_DAA\_Join.

5146 i. The TPM MUST support sufficient resources to perform one (1)  
5147 TPM\_DAA\_Join/ TPM\_DAA\_Sign. The TPM MAY support additional  
5148 TPM\_DAA\_Join/ TPM\_DAA\_Sign sessions.

5149 ii. The TPM may share internal resources between the DAA operations and other  
5150 variable resource requirements:

5151 iii. If there are insufficient resources within the stored key pool (and one or  
5152 more keys need to be removed to permit the DAA operation to execute) return  
5153 TPM\_NOSPACE

5154 iv. If there are insufficient resources within the stored session pool (and  
5155 one or more authorization or transport sessions need to be removed to permit  
5156 the DAA operation to execute), return TPM\_RESOURCES.

5157 d. Set all fields in DAA\_issuerSettings = NULL

5158 e. set all fields in DAA\_tpmSpecific = NULL

5159 f. set all fields in DAA\_session = NULL

5160 g. Set all fields in DAA\_joinSession = NULL

5161 h. Verify that sizeof(inputData0) == sizeof(DAA\_tpmSpecific -> DAA\_count) and return  
5162 error TPM\_DAA\_INPUT\_DATA0 on mismatch

5163 i. Verify that inputData0 > 0, and return error TPM\_DAA\_INPUT\_DATA0 on mismatch  
5164 j. Set DAA\_tpmSpecific -> DAA\_count = inputData0  
5165 k. set DAA\_session -> DAA\_digestContext = SHA-1(DAA\_tpmSpecific ||  
5166 DAA\_joinSession)  
5167 l. set DAA\_session -> DAA\_stage = 1  
5168 m. Assign session handle for TPM\_DAA\_Join  
5169 n. set outputData = new session handle  
5170 i. The handle in outputData is included the output HMAC.  
5171 o. return TPM\_SUCCESS  
51726. If stage==1  
5173 a. Verify that DAA\_session ->DAA\_stage==1. Return TPM\_DAA\_STAGE and flush handle  
5174 on mismatch  
5175 b. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific ||  
5176 DAA\_joinSession) and return TPM\_DAA\_TPM\_SETTINGS on mismatch  
5177 c. Verify that sizeof(inputData0) == DAA\_SIZE\_issuerModulus and return error  
5178 TPM\_DAA\_INPUT\_DATA0 on mismatch  
5179 d. If DAA\_session -> DAA\_scratch == NULL:  
5180 i. Set DAA\_session -> DAA\_scratch = inputData0  
5181 ii. set DAA\_joinSession -> DAA\_digest\_n0 = SHA-1(DAA\_session -> DAA\_scratch)  
5182 iii. set DAA\_tpmSpecific -> DAA\_rekey = SHA-1(tpmDAASeed || DAA\_joinSession  
5183 -> DAA\_digest\_n0)  
5184 e. Else (If DAA\_session -> DAA\_scratch != NULL):  
5185 i. Set signedData = inputData0  
5186 ii. Verify that sizeof(inputData1) == DAA\_SIZE\_issuerModulus and return error  
5187 TPM\_DAA\_INPUT\_DATA1 on mismatch  
5188 iii. Set signatureValue = inputData1  
5189 iv. Use the RSA key == [DAA\_session -> DAA\_scratch] to verify that  
5190 signatureValue is a signature on signedData using  
5191 TPM\_SS\_RSASSAPKCS1v15\_SHA1 (RSA PKCS1.5 with SHA-1), and return error  
5192 TPM\_DAA\_ISSUER\_VALIDITY on mismatch  
5193 v. Set DAA\_session -> DAA\_scratch = signedData  
5194 f. Decrement DAA\_tpmSpecific -> DAA\_count by 1 (unity)  
5195 g. If DAA\_tpmSpecific -> DAA\_count == 0:  
5196 i. increment DAA\_session -> DAA\_stage by 1  
5197 h. set DAA\_session -> DAA\_digestContext = SHA-1(DAA\_tpmSpecific ||  
5198 DAA\_joinSession)  
5199 i. set outputData = NULL



```

1261
1262
5200  j.  return TPM_SUCCESS
52017. If stage==2
5202  a.  Verify that DAA_session ->DAA_stage==2. Return TPM_DAA_STAGE and flush handle
5203      on mismatch
5204  b.  Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5205      DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
5206  c.  Verify that sizeof(inputData0) == sizeof(TPM_DAA_ISSUER) and return error
5207      TPM_DAA_INPUT_DATA0 on mismatch
5208  d.  Set DAA_issuerSettings = inputData0. Verify that all fields in DAA_issuerSettings are
5209      present and return error TPM_DAA_INPUT_DATA0 if not.
5210  e.  Verify that sizeof(inputData1) == DAA_SIZE_issuerModulus and return error
5211      TPM_DAA_INPUT_DATA1 on mismatch
5212  f.  Set signatureValue = inputData1
5213  g.  Set signedData = (DAA_joinSession -> DAA_digest_n0 || DAA_issuerSettings)
5214  h.  Use the RSA key [DAA_session -> DAA_scratch] to verify that signatureValue is a
5215      signature on signedData using TPM_SS_RSASSAPKCS1v15_SHA1 (RSA PKCS1.5 with
5216      SHA-1),, and return error TPM_DAA_ISSUER_VALIDITY on mismatch
5217  i.  Set DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)
5218  j.  set DAA_session -> DAA_digestContext = SHA-1(DAA_tpmSpecific ||
5219      DAA_joinSession)
5220  k.  Set DAA_session -> DAA_scratch = NULL
5221  l.  increment DAA_session -> DAA_stage by 1
5222  m.  return TPM_SUCCESS
52238. If stage==3
5224  a.  Verify that DAA_session ->DAA_stage==3. Return TPM_DAA_STAGE and flush handle
5225      on mismatch
5226  b.  Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5227      return error TPM_DAA_ISSUER_SETTINGS on mismatch
5228  c.  Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5229      DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
5230  d.  Verify that sizeof(inputData0) == sizeof(DAA_tpmSpecific -> DAA_count) and return
5231      error TPM_DAA_INPUT_DATA0 on mismatch
5232  e.  Set DAA_tpmSpecific -> DAA_count = inputData0
5233  f.  Obtain random data from the RNG and store it as DAA_joinSession -> DAA_join_u0
5234  g.  Obtain random data from the RNG and store it as DAA_joinSession -> DAA_join_u1
5235  h.  set outputData = NULL
5236  i.  increment DAA_session -> DAA_stage by 1

```

5237 j. set DAA\_session -> DAA\_digestContext = SHA-1(DAA\_tpmSpecific ||  
5238 DAA\_joinSession)

5239 k. return TPM\_SUCCESS

52409. If stage==4,

5241 a. Verify that DAA\_session ->DAA\_stage==4. Return TPM\_DAA\_STAGE and flush handle  
5242 on mismatch

5243 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5244 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

5245 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific ||  
5246 DAA\_joinSession) and return error TPM\_DAA\_TPM\_SETTINGS on mismatch

5247 d. Set DAA\_generic\_R0 = inputData0

5248 e. Verify that SHA-1(DAA\_generic\_R0) == DAA\_issuerSettings -> DAA\_digest\_R0 and  
5249 return error TPM\_DAA\_INPUT\_DATA0 on mismatch

5250 f. Set DAA\_generic\_n = inputData1

5251 g. Verify that SHA-1(DAA\_generic\_n) == DAA\_issuerSettings -> DAA\_digest\_n and  
5252 return error TPM\_DAA\_INPUT\_DATA1 on mismatch

5253 h. Set X = DAA\_generic\_R0

5254 i. Set n = DAA\_generic\_n

5255 j. Set f = SHA-1(DAA\_tpmSpecific -> DAA\_rekey || DAA\_tpmSpecific -> DAA\_count ||  
5256 0 ) || SHA-1(DAA\_tpmSpecific -> DAA\_rekey || DAA\_tpmSpecific -> DAA\_count || 1 )  
5257 mod DAA\_issuerSettings -> DAA\_generic\_q

5258 k. Set f0 = f mod 2^DAA\_power0 (erase all but the lowest DAA\_power0 bits of f)

5259 l. Set DAA\_session -> DAA\_scratch = (X^f0) mod n

5260 m. set outputData = NULL

5261 n. increment DAA\_session -> DAA\_stage by 1

5262 o. return TPM\_SUCCESS

526310.If stage==5

5264 a. Verify that DAA\_session ->DAA\_stage==5. Return TPM\_DAA\_STAGE and flush handle  
5265 on mismatch

5266 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5267 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

5268 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific ||  
5269 DAA\_joinSession) and return error TPM\_DAA\_TPM\_SETTINGS on mismatch

5270 d. Set DAA\_generic\_R1 = inputData0

5271 e. Verify that SHA-1(DAA\_generic\_R1) == DAA\_issuerSettings -> DAA\_digest\_R1 and  
5272 return error TPM\_DAA\_INPUT\_DATA0 on mismatch

5273 f. Set DAA\_generic\_n = inputData1

1270  
1271

5274 g. Verify that  $\text{SHA-1}(\text{DAA\_generic\_n}) == \text{DAA\_issuerSettings} \rightarrow \text{DAA\_digest\_n}$  and  
5275 return error `TPM_DAA_INPUT_DATA1` on mismatch

5276 h. Set  $X = \text{DAA\_generic\_R1}$

5277 i. Set  $n = \text{DAA\_generic\_n}$

5278 j. Set  $f = \text{SHA-1}(\text{DAA\_tpmSpecific} \rightarrow \text{DAA\_rekey} || \text{DAA\_tpmSpecific} \rightarrow \text{DAA\_count} ||$   
5279  $0) || \text{SHA-1}(\text{DAA\_tpmSpecific} \rightarrow \text{DAA\_rekey} || \text{DAA\_tpmSpecific} \rightarrow \text{DAA\_count} || 1)$   
5280  $\text{mod DAA\_issuerSettings} \rightarrow \text{DAA\_generic\_q}$ .

5281 k. Shift  $f$  right by  $\text{DAA\_power0}$  bits (discard the lowest  $\text{DAA\_power0}$  bits) and label the  
5282 result  $f1$

5283 l. Set  $Z = \text{DAA\_session} \rightarrow \text{DAA\_scratch}$

5284 m. Set  $\text{DAA\_session} \rightarrow \text{DAA\_scratch} = Z \cdot (X^{f1}) \text{ mod } n$

5285 n. set `outputData = NULL`

5286 o. increment  $\text{DAA\_session} \rightarrow \text{DAA\_stage}$  by 1

5287 p. return `TPM_SUCCESS`

5288 11. If  $\text{stage} == 6$

5289 a. Verify that  $\text{DAA\_session} \rightarrow \text{DAA\_stage} == 6$ . Return `TPM_DAA_STAGE` and flush handle  
5290 on mismatch

5291 b. Verify that  $\text{DAA\_tpmSpecific} \rightarrow \text{DAA\_digestIssuer} == \text{SHA-1}(\text{DAA\_issuerSettings})$  and  
5292 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5293 c. Verify that  $\text{DAA\_session} \rightarrow \text{DAA\_digestContext} == \text{SHA-1}(\text{DAA\_tpmSpecific} ||$   
5294  $\text{DAA\_joinSession})$  and return error `TPM_DAA_TPM_SETTINGS` on mismatch

5295 d. Set  $\text{DAA\_generic\_S0} = \text{inputData0}$

5296 e. Verify that  $\text{SHA-1}(\text{DAA\_generic\_S0}) == \text{DAA\_issuerSettings} \rightarrow \text{DAA\_digest\_S0}$  and  
5297 return error `TPM_DAA_INPUT_DATA0` on mismatch

5298 f. Set  $\text{DAA\_generic\_n} = \text{inputData1}$

5299 g. Verify that  $\text{SHA-1}(\text{DAA\_generic\_n}) == \text{DAA\_issuerSettings} \rightarrow \text{DAA\_digest\_n}$  and  
5300 return error `TPM_DAA_INPUT_DATA1` on mismatch

5301 h. Set  $X = \text{DAA\_generic\_S0}$

5302 i. Set  $n = \text{DAA\_generic\_n}$

5303 j. Set  $Z = \text{DAA\_session} \rightarrow \text{DAA\_scratch}$

5304 k. Set  $Y = \text{DAA\_joinSession} \rightarrow \text{DAA\_join\_u0}$

5305 l. Set  $\text{DAA\_session} \rightarrow \text{DAA\_scratch} = Z \cdot (X^Y) \text{ mod } n$

5306 m. set `outputData = NULL`

5307 n. increment  $\text{DAA\_session} \rightarrow \text{DAA\_stage}$  by 1

5308 o. return `TPM_SUCCESS`

5309 12. If  $\text{stage} == 7$

5310 a. Verify that DAA\_session ->DAA\_stage==7. Return TPM\_DAA\_STAGE and flush handle  
5311 on mismatch

5312 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5313 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

5314 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific ||  
5315 DAA\_joinSession) and return error TPM\_DAA\_TPM\_SETTINGS on mismatch

5316 d. Set DAA\_generic\_S1 = inputData0

5317 e. Verify that SHA-1(DAA\_generic\_S1) == DAA\_issuerSettings -> DAA\_digest\_S1 and  
5318 return error TPM\_DAA\_INPUT\_DATA0 on mismatch

5319 f. Set DAA\_generic\_n = inputData1

5320 g. Verify that SHA-1(DAA\_generic\_n) == DAA\_issuerSettings -> DAA\_digest\_n and  
5321 return error TPM\_DAA\_INPUT\_DATA1 on mismatch

5322 h. Set X = DAA\_generic\_S1

5323 i. Set n = DAA\_generic\_n

5324 j. Set Y = DAA\_joinSession -> DAA\_join\_u1

5325 k. Set Z = DAA\_session -> DAA\_scratch

5326 l. Set DAA\_session -> DAA\_scratch =  $Z \cdot (X^Y) \bmod n$

5327 m. Set DAA\_session -> DAA\_digest to the SHA-1 (DAA\_session -> DAA\_scratch ||  
5328 DAA\_tpmSpecific -> DAA\_count || DAA\_joinSession -> DAA\_digest\_n0)

5329 n. set outputData = DAA\_session -> DAA\_scratch

5330 o. set DAA\_session -> DAA\_scratch = NULL

5331 p. increment DAA\_session -> DAA\_stage by 1

5332 q. return TPM\_SUCCESS

5333 13.If stage==8

5334 a. Verify that DAA\_session ->DAA\_stage==8. Return TPM\_DAA\_STAGE and flush handle  
5335 on mismatch

5336 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5337 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

5338 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific ||  
5339 DAA\_joinSession) and return error TPM\_DAA\_TPM\_SETTINGS on mismatch

5340 d. Verify inputSize0 == DAA\_SIZE\_NE and return error TPM\_DAA\_INPUT\_DATA0 on  
5341 mismatch

5342 e. Set NE = decrypt(inputData0, privEK)

5343 f. set outputData = SHA-1(DAA\_session -> DAA\_digest || NE)

5344 g. set DAA\_session -> DAA\_digest = NULL

5345 h. increment DAA\_session -> DAA\_stage by 1

5346 i. return TPM\_SUCCESS

5347 14.If stage==9

- 5348 a. Verify that DAA\_session ->DAA\_stage==9. Return TPM\_DAA\_STAGE and flush handle  
5349 on mismatch
- 5350 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5351 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch
- 5352 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific ||  
5353 DAA\_joinSession) and return error TPM\_DAA\_TPM\_SETTINGS on mismatch
- 5354 d. Set DAA\_generic\_R0 = inputData0
- 5355 e. Verify that SHA-1(DAA\_generic\_R0) == DAA\_issuerSettings -> DAA\_digest\_R0 and  
5356 return error TPM\_DAA\_INPUT\_DATA0 on mismatch
- 5357 f. Set DAA\_generic\_n = inputData1
- 5358 g. Verify that SHA-1(DAA\_generic\_n) == DAA\_issuerSettings -> DAA\_digest\_n and  
5359 return error TPM\_DAA\_INPUT\_DATA1 on mismatch
- 5360 h. Obtain random data from the RNG and store it as DAA\_session -> DAA\_contextSeed
- 5361 i. Obtain DAA\_SIZE\_r0 bytes using the MGF1 function and label them Y. “r0” ||  
5362 DAA\_session -> DAA\_contextSeed is the Z seed.
- 5363 j. Set X = DAA\_generic\_R0
- 5364 k. Set n = DAA\_generic\_n
- 5365 l. Set DAA\_session -> DAA\_scratch = (X^Y) mod n
- 5366 m. set outputData = NULL
- 5367 n. increment DAA\_session -> DAA\_stage by 1
- 5368 o. return TPM\_SUCCESS

5369 15.If stage==10

- 5370 a. Verify that DAA\_session ->DAA\_stage==10. Return TPM\_DAA\_STAGE and flush  
5371 handle on mismatch h
- 5372 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5373 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch
- 5374 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific ||  
5375 DAA\_joinSession) and return error TPM\_DAA\_TPM\_SETTINGS on mismatch
- 5376 d. Set DAA\_generic\_R1 = inputData0
- 5377 e. Verify that SHA-1(DAA\_generic\_R1) == DAA\_issuerSettings -> DAA\_digest\_R1 and  
5378 return error TPM\_DAA\_INPUT\_DATA0 on mismatch
- 5379 f. Set DAA\_generic\_n = inputData1
- 5380 g. Verify that SHA-1(DAA\_generic\_n) == DAA\_issuerSettings -> DAA\_digest\_n and  
5381 return error TPM\_DAA\_INPUT\_DATA1 on mismatch
- 5382 h. Obtain DAA\_SIZE\_r1 bytes using the MGF1 function and label them Y. “r1” ||  
5383 DAA\_session -> DAA\_contextSeed is the Z seed.
- 5384 i. Set X = DAA\_generic\_R1

5385 j. Set  $n = \text{DAA\_generic\_n}$   
5386 k. Set  $Z = \text{DAA\_session} \rightarrow \text{DAA\_scratch}$   
5387 l. Set  $\text{DAA\_session} \rightarrow \text{DAA\_scratch} = Z \cdot (X^Y) \bmod n$   
5388 m. set `outputData` = NULL  
5389 n. increment `DAA_session`  $\rightarrow$  `DAA_stage` by 1  
5390 o. return `TPM_SUCCESS`  
5391 16.If `stage==11`  
5392 a. Verify that `DAA_session`  $\rightarrow$  `DAA_stage==11`. Return `TPM_DAA_STAGE` and flush  
5393 handle on mismatch  
5394 b. Verify that `DAA_tpmSpecific`  $\rightarrow$  `DAA_digestIssuer` == SHA-1(`DAA_issuerSettings`) and  
5395 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch  
5396 c. Verify that `DAA_session`  $\rightarrow$  `DAA_digestContext` == SHA-1(`DAA_tpmSpecific` ||  
5397 `DAA_joinSession`) and return error `TPM_DAA_TPM_SETTINGS` on mismatch  
5398 d. Set `DAA_generic_S0` = `inputData0`  
5399 e. Verify that SHA-1(`DAA_generic_S0`) == `DAA_issuerSettings`  $\rightarrow$  `DAA_digest_S0` and  
5400 return error `TPM_DAA_INPUT_DATA0` on mismatch  
5401 f. Set `DAA_generic_n` = `inputData1`  
5402 g. Verify that SHA-1(`DAA_generic_n`) == `DAA_issuerSettings`  $\rightarrow$  `DAA_digest_n` and  
5403 return error `TPM_DAA_INPUT_DATA1` on mismatch  
5404 h. Obtain `DAA_SIZE_r2` bytes using the MGF1 function and label them Y. “r2” ||  
5405 `DAA_session`  $\rightarrow$  `DAA_contextSeed` is the Z seed.  
5406 i. Set  $X = \text{DAA\_generic\_S0}$   
5407 j. Set  $n = \text{DAA\_generic\_n}$   
5408 k. Set  $Z = \text{DAA\_session} \rightarrow \text{DAA\_scratch}$   
5409 l. Set  $\text{DAA\_session} \rightarrow \text{DAA\_scratch} = Z \cdot (X^Y) \bmod n$   
5410 m. set `outputData` = NULL  
5411 n. increment `DAA_session`  $\rightarrow$  `DAA_stage` by 1  
5412 o. return `TPM_SUCCESS`  
5413 17.If `stage==12`  
5414 a. Verify that `DAA_session`  $\rightarrow$  `DAA_stage==12`. Return `TPM_DAA_STAGE` and flush  
5415 handle on mismatch  
5416 b. Verify that `DAA_tpmSpecific`  $\rightarrow$  `DAA_digestIssuer` == SHA-1(`DAA_issuerSettings`) and  
5417 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch  
5418 c. Verify that `DAA_session`  $\rightarrow$  `DAA_digestContext` == SHA-1(`DAA_tpmSpecific` ||  
5419 `DAA_joinSession`) and return error `TPM_DAA_TPM_SETTINGS` on mismatch  
5420 d. Set `DAA_generic_S1` = `inputData0`

1288  
1289

5421 e. Verify that  $\text{SHA-1}(\text{DAA\_generic\_S1}) == \text{DAA\_issuerSettings} \rightarrow \text{DAA\_digest\_S1}$  and  
5422 return error `TPM_DAA_INPUT_DATA0` on mismatch

5423 f. Set `DAA_generic_n = inputData1`

5424 g. Verify that  $\text{SHA-1}(\text{DAA\_generic\_n}) == \text{DAA\_issuerSettings} \rightarrow \text{DAA\_digest\_n}$  and  
5425 return error `TPM_DAA_INPUT_DATA1` on mismatch

5426 h. Obtain `DAA_SIZE_r3` bytes using the MGF1 function and label them Y. “r3” ||  
5427 `DAA_session` → `DAA_contextSeed` is the Z seed.

5428 i. Set `X = DAA_generic_S1`

5429 j. Set `n = DAA_generic_n`

5430 k. Set `Z = DAA_session` → `DAA_scratch`

5431 l. Set `DAA_session` → `DAA_scratch = Z*(X^Y) mod n`

5432 m. set `outputData = DAA_session` → `DAA_scratch`

5433 n. Set `DAA_session` → `DAA_scratch = NULL`

5434 o. increment `DAA_session` → `DAA_stage` by 1

5435 p. return `TPM_SUCCESS`

5436 18.If `stage==13`

5437 a. Verify that `DAA_session` → `DAA_stage==13`. Return `TPM_DAA_STAGE` and flush  
5438 handle on mismatch

5439 b. Verify that `DAA_tpmSpecific` → `DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and  
5440 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5441 c. Verify that `DAA_session` → `DAA_digestContext == SHA-1(DAA_tpmSpecific ||`  
5442 `DAA_joinSession)` and return error `TPM_DAA_TPM_SETTINGS` on mismatch

5443 d. Set `DAA_generic_gamma = inputData0`

5444 e. Verify that  $\text{SHA-1}(\text{DAA\_generic\_gamma}) == \text{DAA\_issuerSettings} \rightarrow$   
5445 `DAA_digest_gamma` and return error `TPM_DAA_INPUT_DATA0` on mismatch

5446 f. Verify that `inputSize1 == DAA_SIZE_w` and return error `TPM_DAA_INPUT_DATA1` on  
5447 mismatch

5448 g. Set `w = inputData1`

5449 h. Set `w1 = w^( DAA_issuerSettings → DAA_generic_q) mod (DAA_generic_gamma)`

5450 i. If `w1 != 1` (unity), return error `TPM_DAA_WRONG_W`

5451 j. Set `DAA_session` → `DAA_scratch = w`

5452 k. set `outputData = NULL`

5453 l. increment `DAA_session` → `DAA_stage` by 1

5454 m. return `TPM_SUCCESS`.

5455 19.If `stage==14`

5456 a. Verify that `DAA_session` → `DAA_stage==14`. Return `TPM_DAA_STAGE` and flush  
5457 handle on mismatch

5458 b. Verify that  $\text{DAA\_tpmSpecific} \rightarrow \text{DAA\_digestIssuer} == \text{SHA-1}(\text{DAA\_issuerSettings})$  and  
5459 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5460 c. Verify that  $\text{DAA\_session} \rightarrow \text{DAA\_digestContext} == \text{SHA-1}(\text{DAA\_tpmSpecific} ||$   
5461  $\text{DAA\_joinSession})$  and return error `TPM_DAA_TPM_SETTINGS` on mismatch

5462 d. Set  $\text{DAA\_generic\_gamma} = \text{inputData0}$

5463 e. Verify that  $\text{SHA-1}(\text{DAA\_generic\_gamma}) == \text{DAA\_issuerSettings} \rightarrow$   
5464  $\text{DAA\_digest\_gamma}$  and return error `TPM_DAA_INPUT_DATA0` on mismatch

5465 f. Set  $f = \text{SHA-1}(\text{DAA\_tpmSpecific} \rightarrow \text{DAA\_rekey} || \text{DAA\_tpmSpecific} \rightarrow \text{DAA\_count} ||$   
5466  $0) || \text{SHA-1}(\text{DAA\_tpmSpecific} \rightarrow \text{DAA\_rekey} || \text{DAA\_tpmSpecific} \rightarrow \text{DAA\_count} || 1)$   
5467  $\text{mod DAA\_issuerSettings} \rightarrow \text{DAA\_generic\_q}$ .

5468 g. Set  $E = ((\text{DAA\_session} \rightarrow \text{DAA\_scratch})^f) \text{ mod } (\text{DAA\_generic\_gamma})$ .

5469 h. Set  $\text{outputData} = E$

5470 i. increment  $\text{DAA\_session} \rightarrow \text{DAA\_stage}$  by 1

5471 j. return `TPM_SUCCESS`.

547220.If  $\text{stage} == 15$

5473 a. Verify that  $\text{DAA\_session} \rightarrow \text{DAA\_stage} == 15$ . Return `TPM_DAA_STAGE` and flush  
5474 handle on mismatch

5475 b. Verify that  $\text{DAA\_tpmSpecific} \rightarrow \text{DAA\_digestIssuer} == \text{SHA-1}(\text{DAA\_issuerSettings})$  and  
5476 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5477 c. Verify that  $\text{DAA\_session} \rightarrow \text{DAA\_digestContext} == \text{SHA-1}(\text{DAA\_tpmSpecific} ||$   
5478  $\text{DAA\_joinSession})$  and return error `TPM_DAA_TPM_SETTINGS` on mismatch

5479 d. Set  $\text{DAA\_generic\_gamma} = \text{inputData0}$

5480 e. Verify that  $\text{SHA-1}(\text{DAA\_generic\_gamma}) == \text{DAA\_issuerSettings} \rightarrow$   
5481  $\text{DAA\_digest\_gamma}$  and return error `TPM_DAA_INPUT_DATA0` on mismatch

5482 f. Obtain  $\text{DAA\_SIZE\_r0}$  bytes using the MGF1 function and label them  $r0$ . " $r0$ " ||  
5483  $\text{DAA\_session} \rightarrow \text{DAA\_contextSeed}$  is the Z seed.

5484 g. Obtain  $\text{DAA\_SIZE\_r1}$  bytes using the MGF1 function and label them  $r1$ . " $r1$ " ||  
5485  $\text{DAA\_session} \rightarrow \text{DAA\_contextSeed}$  is the Z seed.

5486 h. set  $r = r0 + 2^{\text{DAA\_power0}} * r1 \text{ mod } (\text{DAA\_issuerSettings} \rightarrow \text{DAA\_generic\_q})$ .

5487 i. set  $E1 = ((\text{DAA\_session} \rightarrow \text{DAA\_scratch})^r) \text{ mod } (\text{DAA\_generic\_gamma})$ .

5488 j. Set  $\text{DAA\_session} \rightarrow \text{DAA\_scratch} = \text{NULL}$

5489 k. Set  $\text{outputData} = E1$

5490 l. increment  $\text{DAA\_session} \rightarrow \text{DAA\_stage}$  by 1

5491 m. return `TPM_SUCCESS`.

549221.If  $\text{stage} == 16$

5493 a. Verify that  $\text{DAA\_session} \rightarrow \text{DAA\_stage} == 16$ . Return `TPM_DAA_STAGE` and flush  
5494 handle on mismatch



```

1297
1298
5495 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5496 return error TPM_DAA_ISSUER_SETTINGS on mismatch
5497 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5498 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
5499 d. Verify that inputSize0 == sizeof(TPM_DIGEST) and return error
5500 TPM_DAA_INPUT_DATA0 on mismatch
5501 e. Set DAA_session -> DAA_digest = inputData0
5502 f. Obtain DAA_SIZE_NT bytes from the RNG and label them NT
5503 g. Set DAA_session -> DAA_digest to the SHA-1 ( DAA_session -> DAA_digest || NT )
5504 h. Set outputData = NT
5505 i. increment DAA_session -> DAA_stage by 1
5506 j. return TPM_SUCCESS.
5507 22.If stage==17
5508 a. Verify that DAA_session ->DAA_stage==17. Return TPM_DAA_STAGE and flush
5509 handle on mismatch
5510 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5511 return error TPM_DAA_ISSUER_SETTINGS on mismatch
5512 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5513 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
5514 d. Obtain DAA_SIZE_r0 bytes using the MGF1 function and label them r0. "r0" ||
5515 DAA_session -> DAA_contextSeed is the Z seed.
5516 e. Set f = SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count ||
5517 0 ) || SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1 )
5518 mod DAA_issuerSettings -> DAA_generic_q.
5519 f. Set f0 = f mod 2^DAA_power0 (erase all but the lowest DAA_power0 bits of f)
5520 g. Set s0 = r0 + (DAA_session -> DAA_digest) * f0 in Z. Compute over the integers. The
5521 computation is not reduced with a modulus.
5522 h. set outputData = s0
5523 i. increment DAA_session -> DAA_stage by 1
5524 j. return TPM_SUCCESS
5525 23.If stage==18
5526 a. Verify that DAA_session ->DAA_stage==18. Return TPM_DAA_STAGE and flush
5527 handle on mismatch
5528 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5529 return error TPM_DAA_ISSUER_SETTINGS on mismatch
5530 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5531 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch

```

5532 d. Obtain DAA\_SIZE\_r1 bytes using the MGF1 function and label them r1. “r1” ||  
5533 DAA\_session -> DAA\_contextSeed is the Z seed.

5534 e. Set  $f = \text{SHA-1}(\text{DAA\_tpmSpecific} \rightarrow \text{DAA\_rekey} || \text{DAA\_tpmSpecific} \rightarrow \text{DAA\_count} ||$   
5535  $0) || \text{SHA-1}(\text{DAA\_tpmSpecific} \rightarrow \text{DAA\_rekey} || \text{DAA\_tpmSpecific} \rightarrow \text{DAA\_count} || 1)$   
5536  $\text{mod DAA\_issuerSettings} \rightarrow \text{DAA\_generic\_q}$ .

5537 f. Shift f right by DAA\_power0 bits (discard the lowest DAA\_power0 bits) and label the  
5538 result f1

5539 g. Set  $s1 = r1 + (\text{DAA\_session} \rightarrow \text{DAA\_digest}) * f1$  in **Z**. Compute over the integers. The  
5540 computation is not reduced with a modulus.

5541 h. set outputData = s1

5542 i. increment DAA\_session -> DAA\_stage by 1

5543 j. return TPM\_SUCCESS

554424.If stage==19

5545 a. Verify that DAA\_session ->DAA\_stage==19. Return TPM\_DAA\_STAGE and flush  
5546 handle on mismatch

5547 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5548 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

5549 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific ||  
5550 DAA\_joinSession) and return error TPM\_DAA\_TPM\_SETTINGS on mismatch

5551 d. Obtain DAA\_SIZE\_r2 bytes using the MGF1 function and label them r2. “r2” ||  
5552 DAA\_session -> DAA\_contextSeed is the Z seed.

5553 e. Set  $s2 = r2 + (\text{DAA\_session} \rightarrow \text{DAA\_digest}) * (\text{DAA\_joinSession} \rightarrow \text{DAA\_join\_u0}) \text{ mod}$   
5554  $2^{\text{DAA\_power1}}$  (Erase all but the lowest DAA\_power1 bits of s2)

5555 f. set outputData = s2

5556 g. increment DAA\_session -> DAA\_stage by 1

5557 h. return TPM\_SUCCESS

555825.If stage==20

5559 a. Verify that DAA\_session ->DAA\_stage==20. Return TPM\_DAA\_STAGE and flush  
5560 handle on mismatch

5561 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5562 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

5563 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific ||  
5564 DAA\_joinSession) and return error TPM\_DAA\_TPM\_SETTINGS on mismatch

5565 d. Obtain DAA\_SIZE\_r2 bytes using the MGF1 function and label them r2. “r2” ||  
5566 DAA\_session -> DAA\_contextSeed is the Z seed.

5567 e. Set  $s12 = r2 + (\text{DAA\_session} \rightarrow \text{DAA\_digest}) * (\text{DAA\_joinSession} \rightarrow \text{DAA\_join\_u0})$

5568 f. Shift s12 right by DAA\_power1 bit (discard the lowest DAA\_power1 bits).

5569 g. Set DAA\_session -> DAA\_scratch = s12

```

1306
1307
5570 h. Set outputData = DAA_session -> DAA_digest
5571 i. increment DAA_session -> DAA_stage by 1
5572 j. return TPM_SUCCESS
5573 26.If stage==21
5574 a. Verify that DAA_session ->DAA_stage==21. Return TPM_DAA_STAGE and flush
5575 handle on mismatch
5576 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5577 return error TPM_DAA_ISSUER_SETTINGS on mismatch
5578 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5579 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
5580 d. Obtain DAA_SIZE_r3 bytes using the MGF1 function and label them r3. "r3" ||
5581 DAA_session -> DAA_contextSeed is the Z seed.
5582 e. Set s3 = r3 + (DAA_session -> DAA_digest)*( DAA_joinSession -> DAA_join_u1) +
5583 (DAA_session -> DAA_scratch).
5584 f. Set DAA_session -> DAA_scratch = NULL
5585 g. set outputData = s3
5586 h. increment DAA_session -> DAA_stage by 1
5587 i. return TPM_SUCCESS
5588 27.If stage==22
5589 a. Verify that DAA_session ->DAA_stage==22. Return TPM_DAA_STAGE and flush
5590 handle on mismatch
5591 b. Verify that DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings) and
5592 return error TPM_DAA_ISSUER_SETTINGS on mismatch
5593 c. Verify that DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific ||
5594 DAA_joinSession) and return error TPM_DAA_TPM_SETTINGS on mismatch
5595 d. Verify inputSize0 == DAA_SIZE_v0 and return error TPM_DAA_INPUT_DATA0 on
5596 mismatch
5597 e. Set u2 = inputData0
5598 f. Set v0 = u2 + (DAA_joinSession -> DAA_join_u0) mod 2^DAA_power1 (Erase all but
5599 the lowest DAA_power1 bits of v0).
5600 g. Set DAA_tpmSpecific -> DAA_digest_v0 = SHA-1(v0)
5601 h. Set v10 = u2 + (DAA_joinSession -> DAA_join_u0) in Z. Compute over the integers.
5602 The computation is not reduced with a modulus.
5603 i. Shift v10 right by DAA_power1 bits (erase the lowest DAA_power1 bits).
5604 j. Set DAA_session ->DAA_scratch = v10
5605 k. Set outputData
5606 i. Fill in TPM_DAA_BLOB with a type of TPM_RT_DAA_V0 and encrypt the v0
5607 parameters using TPM_PERMANENT_DATA -> daaBlobKey

```

5608           ii. set outputData to the encrypted TPM\_DAA\_BLOB

5609    1. increment DAA\_session -> DAA\_stage by 1

5610    m. set    DAA\_session   ->   DAA\_digestContext   =   SHA-1(DAA\_tpmSpecific   ||  
5611    DAA\_joinSession)

5612    n. return TPM\_SUCCESS

561328.If stage==23

5614    a. Verify that DAA\_session ->DAA\_stage==23. Return TPM\_DAA\_STAGE and flush  
5615    handle on mismatch

5616    b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5617    return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

5618    c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific ||  
5619    DAA\_joinSession) and return error TPM\_DAA\_TPM\_SETTINGS on mismatch

5620    d. Verify inputSize0 == DAA\_SIZE\_v1 and return error TPM\_DAA\_INPUT\_DATA0 on  
5621    mismatch

5622    e. Set u3 = inputData0

5623    f. Set v1 = u3 + DAA\_joinSession -> DAA\_join\_u1 + DAA\_session ->DAA\_scratch

5624    g. Set DAA\_tpmSpecific -> DAA\_digest\_v1 = SHA-1(v1)

5625    h. Set outputData

5626           i. Fill in TPM\_DAA\_BLOB with a type of TPM\_RT\_DAA\_V1 and encrypt the v1  
5627           parameters using TPM\_PERMANENT\_DATA -> daaBlobKey

5628           ii. set outputData to the encrypted TPM\_DAA\_BLOB

5629    i. Set DAA\_session ->DAA\_scratch = NULL

5630    j. increment DAA\_session -> DAA\_stage by 1

5631    k. set    DAA\_session   ->   DAA\_digestContext   =   SHA-1(DAA\_tpmSpecific   ||  
5632    DAA\_joinSession)

5633    l. return TPM\_SUCCESS

563429.If stage==24

5635    a. Verify that DAA\_session ->DAA\_stage==24. Return TPM\_DAA\_STAGE and flush  
5636    handle on mismatch

5637    b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5638    return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

5639    c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific ||  
5640    DAA\_joinSession) and return error TPM\_DAA\_TPM\_SETTINGS on mismatch

5641    d. set    outputData   =   enc(DAA\_tpmSpecific)   using   TPM\_PERMANENT\_DATA   ->  
5642    daaBlobKey

5643    e. Terminate the DAA session and all resources associated with the DAA join session  
5644    handle.

5645    f. return TPM\_SUCCESS

564630.If stage > 24, return error: TPM\_DAA\_STAGE

## 5647**26.2 TPM\_DAA\_Sign**

### 5648**Start of informative comment:**

5649outputSize and outputData are always included in the outParamDigest. This includes stage  
56500, where the outputData contains the DAA session handle.

### 5651**End of informative comment.**

5652TPM protected capability; user must provide authorizations from the TPM Owner.

### 5653**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.

### 5654**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.

5655**Description**

5656This table summaries the input, output and saved data that is associated with each stage of  
5657processing.

Stage	Input Data0	Input Data1	Operation	Output Data	Scratchpad
0	DAA_issuerSettings	NULL	initialise	handle	NULL
1	enc(DAA_tpmSpecific)	NULL	initialise	NULL	NULL
2	DAA_generic_R0	DAA_generic_n	$P1 = R0^{r0} \bmod n$	NULL	P1
3	DAA_generic_R1	DAA_generic_n	$P2 = P1 * (R1^{r1}) \bmod n$	NULL	P2
4	DAA_generic_S0	DAA_generic_n	$P3 = P2 * (S0^{r2}) \bmod n$	NULL	P3
5	DAA_generic_S1	DAA_generic_n	$T = P3 * (S1^{r4}) \bmod n$	T	NULL
6	DAA_generic_gamma	w	$w1 = w^q \bmod \text{gamma}$	NULL	w
7	DAA_generic_gamma	NULL	$E = w^f \bmod \text{gamma}$	E	w
8	DAA_generic_gamma	NULL	$r = r0 + (2^{\text{power0}}) * r1 \bmod q,$ $E1 = w^r \bmod \text{gamma}$	E1	NULL
9	c1	NULL	$c = \text{hash}(c1 \parallel NT)$	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	$s2 = r2 + c * v0 \bmod 2^{\text{power1}}$	s2	NULL
14	enc(v0)	NULL	$s12 = r2 + c * v0 \gg \text{power1}$	NULL	s12
15	enc(v1)	NULL	$s3 = r4 + c * v1 + s12$	s3	NULL

5658

5659When a TPM receives an Owner authorized command to input enc(DAA\_tpmSpecific) or  
5660enc(v0) or enc(v1), the TPM MUST verify that the TPM created the data and that neither the  
5661data nor the TPM's daaProof has been changed since the data was created. Loading one of  
5662these wrapped blobs does not require authorization, since correct blobs were created by the  
5663TPM under Owner authorization, and unwrapped blobs cannot be used without Owner  
5664authorisation. The TPM MUST NOT restrict the number of times that the contents of  
5665enc(DAA\_tpmSpecific) or enc(v0) or enc(v1) can be used by the same combination of TPM  
5666and daaProof that created them.

5667**Actions**

5668A Trusted Platform Module that receives a valid TPM\_DAA\_Sign command SHALL:

56691. Use ownerAuth to verify that the Owner authorized all TPM\_DAA\_Sign input parameters.
56702. Any error results in the TPM invalidating all resources associated with the command
56713. Constant values of 0 or 1 are 1 byte integers, stages affected are
- 5672 a. 7(f), 11(e), 12(e)

56734. Representation of the strings “r0” to “r4” are 2-byte ASCII encodings, stages affected are  
5674 a. 2(h), 3(h), 4(h), 5(h), 12(d), 13(f), 14(f), 15(f)

## 5675Stages

56760. If stage==0

- 5677 b. Determine that sufficient resources are available to perform a TPM\_DAA\_Sign.
- 5678 i. The TPM MUST support sufficient resources to perform one (1)  
5679 TPM\_DAA\_Join/ TPM\_DAA\_Sign. The TPM MAY support addition TPM\_DAA\_Join/  
5680 TPM\_DAA\_Sign sessions.
- 5681 ii. The TPM may share internal resources between the DAA operations and other  
5682 variable resource requirements:
- 5683 iii. If there are insufficient resources within the stored key pool (and one or more  
5684 keys need to be removed to permit the DAA operation to execute) return  
5685 TPM\_NOSPACE
- 5686 iv. If there are insufficient resources within the stored session pool (and one or  
5687 more authorization or transport sessions need to be removed to permit the DAA  
5688 operation to execute), return TPM\_RESOURCES.
- 5689 c. Set DAA\_issuerSettings = inputData0
- 5690 d. Verify that all fields in DAA\_issuerSettings are present and return error  
5691 TPM\_DAA\_INPUT\_DATA0 if not.
- 5692 e. set all fields in DAA\_session = NULL
- 5693 f. Assign new handle for session
- 5694 g. Set outputData to new handle
- 5695 i. The handle in outputData is included the output HMAC.
- 5696 h. set DAA\_session -> DAA\_stage = 1
- 5697 i. return TPM\_SUCCESS

56985. If stage==1

- 5699 a. Verify that DAA\_session ->DAA\_stage==1. Return TPM\_DAA\_STAGE and flush handle  
5700 on mismatch
- 5701 b. Set DAA\_tpmSpecific = unwrap(inputData0) using TPM\_PERMANENT\_DATA ->  
5702 daaBlobKey
- 5703 c. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5704 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch
- 5705 d. set DAA\_session -> DAA\_digestContext = SHA-1(DAA\_tpmSpecific)
- 5706 e. set outputData = NULL
- 5707 f. set DAA\_session -> DAA\_stage =2
- 5708 g. return TPM\_SUCCESS

57096. If stage==2



1333  
1334

- 5710 a. Verify that DAA\_session -> DAA\_stage==2. Return TPM\_DAA\_STAGE and flush handle  
5711 on mismatch
- 5712 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5713 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch
- 5714 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific) and  
5715 return error TPM\_DAA\_TPM\_SETTINGS on mismatch
- 5716 d. Set DAA\_generic\_R0 = inputData0
- 5717 e. Verify that SHA-1(DAA\_generic\_R0) == DAA\_issuerSettings -> DAA\_digest\_R0 and  
5718 return error TPM\_DAA\_INPUT\_DATA0 on mismatch
- 5719 f. Set DAA\_generic\_n = inputData1
- 5720 g. Verify that SHA-1(DAA\_generic\_n) == DAA\_issuerSettings -> DAA\_digest\_n and  
5721 return error TPM\_DAA\_INPUT\_DATA1 on mismatch
- 5722 h. Obtain random data from the RNG and store it as DAA\_session -> DAA\_contextSeed
- 5723 i. Obtain DAA\_SIZE\_r0 bytes using the MGF1 function and label them Y. “r0” ||  
5724 DAA\_session -> DAA\_contextSeed is the Z seed.
- 5725 j. Set X = DAA\_generic\_R0
- 5726 k. Set n = DAA\_generic\_n
- 5727 l. Set DAA\_session -> DAA\_scratch = (X^Y) mod n
- 5728 m. set outputData = NULL
- 5729 n. increment DAA\_session -> DAA\_stage by 1
- 5730 o. return TPM\_SUCCESS
- 5731 17. If stage==3
- 5732 a. Verify that DAA\_session -> DAA\_stage==3. Return TPM\_DAA\_STAGE and flush handle  
5733 on mismatch
- 5734 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5735 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch
- 5736 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific) and  
5737 return error TPM\_DAA\_TPM\_SETTINGS on mismatch
- 5738 d. Set DAA\_generic\_R1 = inputData0
- 5739 e. Verify that SHA-1(DAA\_generic\_R1) == DAA\_issuerSettings -> DAA\_digest\_R1 and  
5740 return error TPM\_DAA\_INPUT\_DATA0 on mismatch
- 5741 f. Set DAA\_generic\_n = inputData1
- 5742 g. Verify that SHA-1(DAA\_generic\_n) == DAA\_issuerSettings -> DAA\_digest\_n and  
5743 return error TPM\_DAA\_INPUT\_DATA1 on mismatch
- 5744 h. Obtain DAA\_SIZE\_r1 bytes using the MGF1 function and label them Y. “r1” ||  
5745 DAA\_session -> DAA\_contextSeed is the Z seed.
- 5746 i. Set X = DAA\_generic\_R1
- 5747 j. Set n = DAA\_generic\_n

5748 k. Set  $Z = \text{DAA\_session} \rightarrow \text{DAA\_scratch}$   
5749 l. Set  $\text{DAA\_session} \rightarrow \text{DAA\_scratch} = Z \cdot (X^Y) \bmod n$   
5750 m. set `outputData = NULL`  
5751 n. increment `DAA_session`  $\rightarrow$  `DAA_stage` by 1  
5752 o. return `TPM_SUCCESS`  
57538. If `stage==4`  
5754 a. Verify that `DAA_session`  $\rightarrow$  `DAA_stage==4`. Return `TPM_DAA_STAGE` and flush handle  
5755 on mismatch  
5756 b. Verify that `DAA_tpmSpecific`  $\rightarrow$  `DAA_digestIssuer` == `SHA-1(DAA_issuerSettings)` and  
5757 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch  
5758 c. Verify that `DAA_session`  $\rightarrow$  `DAA_digestContext` = `SHA-1(DAA_tpmSpecific)` and  
5759 return error `TPM_DAA_TPM_SETTINGS` on mismatch  
5760 d. Set `DAA_generic_S0` = `inputData0`  
5761 e. Verify that `SHA-1(DAA_generic_S0)` == `DAA_issuerSettings`  $\rightarrow$  `DAA_digest_S0` and  
5762 return error `TPM_DAA_INPUT_DATA0` on mismatch  
5763 f. Set `DAA_generic_n` = `inputData1`  
5764 g. Verify that `SHA-1(DAA_generic_n)` == `DAA_issuerSettings`  $\rightarrow$  `DAA_digest_n` and  
5765 return error `TPM_DAA_INPUT_DATA1` on mismatch  
5766 h. Obtain `DAA_SIZE_r2` bytes using the MGF1 function and label them Y. “r2” ||  
5767 `DAA_session`  $\rightarrow$  `DAA_contextSeed` is the Z seed.  
5768 i. Set  $X = \text{DAA\_generic\_S0}$   
5769 j. Set  $n = \text{DAA\_generic\_n}$   
5770 k. Set  $Z = \text{DAA\_session} \rightarrow \text{DAA\_scratch}$   
5771 l. Set  $\text{DAA\_session} \rightarrow \text{DAA\_scratch} = Z \cdot (X^Y) \bmod n$   
5772 m. set `outputData = NULL`  
5773 n. increment `DAA_session`  $\rightarrow$  `DAA_stage` by 1  
5774 o. return `TPM_SUCCESS`  
57759. If `stage==5`  
5776 a. Verify that `DAA_session`  $\rightarrow$  `DAA_stage==5`. Return `TPM_DAA_STAGE` and flush handle  
5777 on mismatch  
5778 b. Verify that `DAA_tpmSpecific`  $\rightarrow$  `DAA_digestIssuer` == `SHA-1(DAA_issuerSettings)` and  
5779 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch  
5780 c. Verify that `DAA_session`  $\rightarrow$  `DAA_digestContext` == `SHA-1(DAA_tpmSpecific)` and  
5781 return error `TPM_DAA_TPM_SETTINGS` on mismatch  
5782 d. Set `DAA_generic_S1` = `inputData0`  
5783 e. Verify that `SHA-1(DAA_generic_S1)` == `DAA_issuerSettings`  $\rightarrow$  `DAA_digest_S1` and  
5784 return error `TPM_DAA_INPUT_DATA0` on mismatch

1342  
1343

5785 f. Set DAA\_generic\_n = inputData1

5786 g. Verify that  $\text{SHA-1}(\text{DAA\_generic\_n}) == \text{DAA\_issuerSettings} \rightarrow \text{DAA\_digest\_n}$  and  
5787 return error TPM\_DAA\_INPUT\_DATA1 on mismatch

5788 h. Obtain DAA\_SIZE\_r4 bytes using the MGF1 function and label them Y. “r4” ||  
5789 DAA\_session -> DAA\_contextSeed is the Z seed.

5790 i. Set X = DAA\_generic\_S1

5791 j. Set n = DAA\_generic\_n

5792 k. Set Z = DAA\_session -> DAA\_scratch

5793 l. Set DAA\_session -> DAA\_scratch =  $Z * (X^Y) \bmod n$

5794 m. set outputData = DAA\_session -> DAA\_scratch

5795 n. set DAA\_session -> DAA\_scratch = NULL

5796 o. increment DAA\_session -> DAA\_stage by 1

5797 p. return TPM\_SUCCESS

5798 10. If stage==6

5799 a. Verify that DAA\_session -> DAA\_stage==6. Return TPM\_DAA\_STAGE and flush handle  
5800 on mismatch

5801 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer ==  $\text{SHA-1}(\text{DAA\_issuerSettings})$  and  
5802 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

5803 c. Verify that DAA\_session -> DAA\_digestContext ==  $\text{SHA-1}(\text{DAA\_tpmSpecific})$  and  
5804 return error TPM\_DAA\_TPM\_SETTINGS on mismatch

5805 d. Set DAA\_generic\_gamma = inputData0

5806 e. Verify that  $\text{SHA-1}(\text{DAA\_generic\_gamma}) == \text{DAA\_issuerSettings} \rightarrow$   
5807 DAA\_digest\_gamma and return error TPM\_DAA\_INPUT\_DATA0 on mismatch

5808 f. Verify that inputSize1 == DAA\_SIZE\_w and return error TPM\_DAA\_INPUT\_DATA1 on  
5809 mismatch

5810 g. Set w = inputData1

5811 h. Set  $w1 = w^{\wedge}(\text{DAA\_issuerSettings} \rightarrow \text{DAA\_generic\_q}) \bmod (\text{DAA\_generic\_gamma})$

5812 i. If  $w1 \neq 1$  (unity), return error TPM\_DAA\_WRONG\_W

5813 j. Set DAA\_session -> DAA\_scratch = w

5814 k. set outputData = NULL

5815 l. increment DAA\_session -> DAA\_stage by 1

5816 m. return TPM\_SUCCESS.

5817 11. If stage==7

5818 a. Verify that DAA\_session -> DAA\_stage==7. Return TPM\_DAA\_STAGE and flush handle  
5819 on mismatch

5820 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer ==  $\text{SHA-1}(\text{DAA\_issuerSettings})$  and  
5821 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

5822 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific) and  
5823 return error TPM\_DAA\_TPM\_SETTINGS on mismatch

5824 d. Set DAA\_generic\_gamma = inputData0

5825 e. Verify that SHA-1(DAA\_generic\_gamma) == DAA\_issuerSettings ->  
5826 DAA\_digest\_gamma and return error TPM\_DAA\_INPUT\_DATA0 on mismatch

5827 f. Set  $f = \text{SHA-1}(\text{DAA\_tpmSpecific} \rightarrow \text{DAA\_rekey} || \text{DAA\_tpmSpecific} \rightarrow \text{DAA\_count} ||$   
5828  $0) || \text{SHA-1}(\text{DAA\_tpmSpecific} \rightarrow \text{DAA\_rekey} || \text{DAA\_tpmSpecific} \rightarrow \text{DAA\_count} || 1)$   
5829  $\text{mod DAA\_issuerSettings} \rightarrow \text{DAA\_generic\_q}.$

5830 g. Set  $E = ((\text{DAA\_session} \rightarrow \text{DAA\_scratch})^f) \text{ mod } (\text{DAA\_generic\_gamma}).$

5831 h. Set outputData = E

5832 i. increment DAA\_session -> DAA\_stage by 1

5833 j. return TPM\_SUCCESS.

5834 12.If stage==8

5835 a. Verify that DAA\_session ->DAA\_stage==8. Return TPM\_DAA\_STAGE and flush handle  
5836 on mismatch

5837 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5838 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

5839 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific) and  
5840 return error TPM\_DAA\_TPM\_SETTINGS on mismatch

5841 d. Set DAA\_generic\_gamma = inputData0

5842 e. Verify that SHA-1(DAA\_generic\_gamma) == DAA\_issuerSettings ->  
5843 DAA\_digest\_gamma and return error TPM\_DAA\_INPUT\_DATA0 on mismatch

5844 f. Obtain DAA\_SIZE\_r0 bytes using the MGF1 function and label them r0. “r0” ||  
5845 DAA\_session -> DAA\_contextSeed is the Z seed.

5846 g. Obtain DAA\_SIZE\_r1 bytes using the MGF1 function and label them r1. “r1” ||  
5847 DAA\_session -> DAA\_contextSeed is the Z seed.

5848 h. set  $r = r0 + 2^{\text{DAA\_power0}} * r1 \text{ mod } (\text{DAA\_issuerSettings} \rightarrow \text{DAA\_generic\_q}).$

5849 i. Set  $E1 = ((\text{DAA\_session} \rightarrow \text{DAA\_scratch})^r) \text{ mod } (\text{DAA\_generic\_gamma})$

5850 j. Set DAA\_session -> DAA\_scratch = NULL

5851 k. Set outputData = E1

5852 l. increment DAA\_session -> DAA\_stage by 1

5853 m. return TPM\_SUCCESS.

5854 13.If stage==9

5855 a. Verify that DAA\_session ->DAA\_stage==9. Return TPM\_DAA\_STAGE and flush handle  
5856 on mismatch

5857 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5858 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

1351  
1352

5859 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific) and  
5860 return error TPM\_DAA\_TPM\_SETTINGS on mismatch

5861 d. Verify that inputSize0 == sizeof(TPM\_DIGEST) and return error  
5862 TPM\_DAA\_INPUT\_DATA0 on mismatch

5863 e. Set DAA\_session -> DAA\_digest = inputData0

5864 f. Obtain DAA\_SIZE\_NT bytes from the RNG and label them NT

5865 g. Set DAA\_session -> DAA\_digest to the SHA-1 ( DAA\_session -> DAA\_digest || NT )

5866 h. Set outputData = NT

5867 i. increment DAA\_session -> DAA\_stage by 1

5868 j. return TPM\_SUCCESS.

5869 14.If stage==10

5870 a. Verify that DAA\_session ->DAA\_stage==10. Return TPM\_DAA\_STAGE and flush  
5871 handle on mismatch

5872 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5873 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

5874 c. Verify that DAA\_session -> DAA\_digestContext == SHA-1(DAA\_tpmSpecific) and  
5875 return error TPM\_DAA\_TPM\_SETTINGS on mismatch

5876 d. Verify that inputSize0 == sizeof(BYTE), and return error TPM\_DAA\_INPUT\_DATA0 on  
5877 mismatch

5878 e. Set selector = inputData0, verify that selector == 0 or 1, and return error  
5879 TPM\_DAA\_INPUT\_DATA0 on mismatch

5880 f. If selector == 1, verify that inputSize1 == sizeof(TPM\_DIGEST), and return error  
5881 TPM\_DAA\_INPUT\_DATA1 on mismatch

5882 g. Set DAA\_session -> DAA\_digest to SHA-1 (DAA\_session -> DAA\_digest || 1 ||  
5883 inputData1)

5884 h. If selector == 0, verify that inputData1 is a handle to a TPM identity key (AIK), and  
5885 return error TPM\_DAA\_INPUT\_DATA1 on mismatch

5886 i. Set DAA\_session -> DAA\_digest to SHA-1 (DAA\_session -> DAA\_digest || 0 || n2)  
5887 where n2 is the modulus of the AIK

5888 j. Set outputData = DAA\_session -> DAA\_digest

5889 k. increment DAA\_session -> DAA\_stage by 1

5890 l. return TPM\_SUCCESS.

5891 15.If stage==11

5892 a. Verify that DAA\_session ->DAA\_stage==11. Return TPM\_DAA\_STAGE and flush  
5893 handle on mismatch

5894 b. Verify that DAA\_tpmSpecific -> DAA\_digestIssuer == SHA-1(DAA\_issuerSettings) and  
5895 return error TPM\_DAA\_ISSUER\_SETTINGS on mismatch

5896 c. Verify that `DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific)` and  
5897 return error `TPM_DAA_TPM_SETTINGS` on mismatch

5898 d. Obtain `DAA_SIZE_r0` bytes using the MGF1 function and label them `r0`. "`r0`" ||  
5899 `DAA_session -> DAA_contextSeed` is the Z seed.

5900 e. Set `f = SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count ||`  
5901 `0) || SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1)`  
5902 `mod DAA_issuerSettings -> DAA_generic_q`.

5903 f. Set `f0 = f mod 2^DAA_power0` (erase all but the lowest `DAA_power0` bits of `f`)

5904 g. Set `s0 = r0 + (DAA_session -> DAA_digest)*(f0)`

5905 h. set `outputData = s0`

5906 i. increment `DAA_session -> DAA_stage` by 1

5907 j. return `TPM_SUCCESS`

5908 16.If `stage==12`

5909 a. Verify that `DAA_session ->DAA_stage==12`. Return `TPM_DAA_STAGE` and flush  
5910 handle on mismatch

5911 b. Verify that `DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and  
5912 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5913 c. Verify that `DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific)` and  
5914 return error `TPM_DAA_TPM_SETTINGS` on mismatch

5915 d. Obtain `DAA_SIZE_r1` bytes using the MGF1 function and label them `r1`. "`r1`" ||  
5916 `DAA_session -> DAA_contextSeed` is the Z seed.

5917 e. Set `f = SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count ||`  
5918 `0) || SHA-1(DAA_tpmSpecific -> DAA_rekey || DAA_tpmSpecific -> DAA_count || 1)`  
5919 `mod DAA_issuerSettings -> DAA_generic_q`.

5920 f. Shift `f` right by `DAA_power0` bits (discard the lowest `DAA_power0` bits) and label the  
5921 result `f1`

5922 g. Set `s1 = r1 + (DAA_session -> DAA_digest)*(f1)`

5923 h. set `outputData = s1`

5924 i. increment `DAA_session -> DAA_stage` by 1

5925 j. return `TPM_SUCCESS`

5926 17.If `stage==13`

5927 a. Verify that `DAA_session ->DAA_stage==13`. Return `TPM_DAA_STAGE` and flush  
5928 handle on mismatch

5929 b. Verify that `DAA_tpmSpecific -> DAA_digestIssuer == SHA-1(DAA_issuerSettings)` and  
5930 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5931 c. Verify that `DAA_session -> DAA_digestContext == SHA-1(DAA_tpmSpecific)` and  
5932 return error `TPM_DAA_TPM_SETTINGS` on mismatch

5933 d. Set `DAA_private_v0= unwrap(inputData0) using TPM_PERMANENT_DATA ->`  
5934 `daaBlobKey`

1360  
1361

5935 e. Verify that  $\text{SHA-1}(\text{DAA\_private\_v0}) == \text{DAA\_tpmSpecific} \rightarrow \text{DAA\_digest\_v0}$  and return  
5936 error `TPM_DAA_INPUT_DATA0` on mismatch

5937 f. Obtain `DAA_SIZE_r2` bytes from the MGF1 function and label them `r2`. “`r2`” ||  
5938 `DAA_session` -> `DAA_contextSeed` is the Z seed.

5939 g. Set  $s2 = r2 + (\text{DAA\_session} \rightarrow \text{DAA\_digest}) * (\text{DAA\_private\_v0}) \bmod 2^{\text{DAA\_power1}}$   
5940 (erase all but the lowest `DAA_power1` bits of `s2`)

5941 h. set `outputData = s2`

5942 i. increment `DAA_session` -> `DAA_stage` by 1

5943 j. return `TPM_SUCCESS`

5944 18.If `stage==14`

5945 a. Verify that `DAA_session` -> `DAA_stage==14`. Return `TPM_DAA_STAGE` and flush  
5946 handle on mismatch

5947 b. Verify that `DAA_tpmSpecific` -> `DAA_digestIssuer` ==  $\text{SHA-1}(\text{DAA\_issuerSettings})$  and  
5948 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5949 c. Verify that `DAA_session` -> `DAA_digestContext` ==  $\text{SHA-1}(\text{DAA\_tpmSpecific})$  and  
5950 return error `TPM_DAA_TPM_SETTINGS` on mismatch

5951 d. Set `DAA_private_v0` = `unwrap(inputData0)` using `TPM_PERMANENT_DATA` ->  
5952 `daaBlobKey`

5953 e. Verify that  $\text{SHA-1}(\text{DAA\_private\_v0}) == \text{DAA\_tpmSpecific} \rightarrow \text{DAA\_digest\_v0}$  and return  
5954 error `TPM_DAA_INPUT_DATA0` on mismatch

5955 f. Obtain `DAA_SIZE_r2` bytes using the MGF1 function and label them `r2`. “`r2`” ||  
5956 `DAA_session` -> `DAA_contextSeed` is the Z seed.

5957 g. Set  $s12 = r2 + (\text{DAA\_session} \rightarrow \text{DAA\_digest}) * (\text{DAA\_private\_v0})$ .

5958 h. Shift `s12` right by `DAA_power1` bits (erase the lowest `DAA_power1` bits).

5959 i. Set `DAA_session` -> `DAA_scratch` = `s12`

5960 j. set `outputData = NULL`

5961 k. increment `DAA_session` -> `DAA_stage` by 1

5962 l. return `TPM_SUCCESS`

5963 19.If `stage==15`

5964 a. Verify that `DAA_session` -> `DAA_stage==15`. Return `TPM_DAA_STAGE` and flush  
5965 handle on mismatch

5966 b. Verify that `DAA_tpmSpecific` -> `DAA_digestIssuer` ==  $\text{SHA-1}(\text{DAA\_issuerSettings})$  and  
5967 return error `TPM_DAA_ISSUER_SETTINGS` on mismatch

5968 c. Verify that `DAA_session` -> `DAA_digestContext` ==  $\text{SHA-1}(\text{DAA\_tpmSpecific})$  and  
5969 return error `TPM_DAA_TPM_SETTINGS` on mismatch

5970 d. Set `DAA_private_v1` = `unwrap(inputData0)` using `TPM_PERMANENT_DATA` ->  
5971 `daaBlobKey`

5972 e. Verify that  $\text{SHA-1}(\text{DAA\_private\_v1}) == \text{DAA\_tpmSpecific} \rightarrow \text{DAA\_digest\_v1}$  and return  
5973 error `TPM_DAA_INPUT_DATA0` on mismatch

5974 f. Obtain `DAA_SIZE_r4` bytes using the MGF1 function and label them `r4`. “`r4`” ||  
5975 `DAA_session`  $\rightarrow$  `DAA_contextSeed` is the Z seed.

5976 g. Set `s3` = `r4` + (`DAA_session`  $\rightarrow$  `DAA_digest`)\*(`DAA_private_v1`) + (`DAA_session`  $\rightarrow$   
5977 `DAA_scratch`).

5978 h. Set `DAA_session`  $\rightarrow$  `DAA_scratch` = NULL

5979 i. set `outputData` = `s3`

5980 j. Terminate the DAA session and all resources associated with the DAA sign session  
5981 handle.

5982 k. return `TPM_SUCCESS`

598320.If stage > 15, return error: `TPM_DAA_STAGE`



## 27. Deprecated commands

### **Start of informative comment:**

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

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

### **End of informative comment.**

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

2. Commands deprecated in version 1.2 **MAY** work with version 1.2 structures

## 599327.1 Key commands

### 5994Start of informative comment:

5995The key commands are deprecated as the new way to handle keys is to use the standard  
5996context commands. So TPM\_EvictKey is now handled by TPM\_FlushSpecific,  
5997TPM\_Terminate\_Handle by TPM\_FlushSpecific.

### 5998End of informative comment.

## 599927.1.1 TPM\_EvictKey

### 6000Incoming 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.

### 6001Outgoing 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

### 6002Actions

6003The TPM will invalidate the key stored in the specified handle and return the space to the  
6004available internal pool for subsequent query by TPM\_GetCapability and usage by  
6005TPM\_LoadKey. If the specified key handle does not correspond to a valid key, an error will  
6006be returned.

### 6007New 1.2 functionality

6008The command must check the status of the ownerEvict flag for the key and if the flag is  
6009TRUE return TPM\_KEY\_CONTROL\_OWNER

## 6010**27.1.2 TPM\_Terminate\_Handle**

### 6011**Start of informative comment:**

6012This allows the TPM manager to clear out information in a session handle.

6013The TPM may maintain the authorization session even though a key attached to it has been  
6014unloaded or the authorization session itself has been unloaded in some way. When a  
6015command is executed that requires this session, it is the responsibility of the external  
6016software to load both the entity and the authorization session information prior to  
6017command execution.

### 6018**End of informative comment.**

### 6019**Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_Terminate_Handle.
4	4			TPM_AUTHHANDLE	handle	The handle to terminate

### 6020**Outgoing Operands and Sizes**

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

### 6021**Description**

6022The TPM SHALL terminate the session and destroy all data associated with the session  
6023indicated.

### 6024**Actions**

6025A TPM SHALL unilaterally perform the actions of TPM\_Terminate\_Handle upon detection of  
6026the following events:

60271. Completion of a received command whose authorization “continueUse” flag is FALSE.
60282. Completion of a received command when a shared secret derived from the authorization  
6029 session was exclusive-or’ed with data (to provide confidentiality for that data). This  
6030 occurs during execution of a TPM\_ChangeAuth command, for example.
60313. When the associated entity is destroyed (in the case of TPM Owner or SRK, for example)
60324. Upon execution of TPM\_Init

60335. When the command returns an error. This is due to the fact that when returning an  
6034 error the TPM does not send back nonceEven. There is no way to maintain the rolling  
6035 nonces, hence the TPM MUST terminate the authorization session.
60366. Failure of an authorization check belonging to that authorization session.

## 603727.2 Context management

### 6038Start of informative comment:

6039The 1.1 context commands were written for specific resource types. The 1.2 commands are  
6040generic for all resource types. So the Savexxx commands are replaced by TPM\_SaveContext  
6041and the LoadXXX commands by TPM\_LoadContext.

### 6042End of informative comment.

## 604327.2.1 TPM\_SaveKeyContext

### 6044Start of informative comment:

6045TPM\_SaveKeyContext saves a loaded key outside the TPM. After creation of the key context  
6046blob the TPM automatically releases the internal memory used by that key. The format of  
6047the key context blob is specific to a TPM.

### 6048End of informative comment.

### 6049Incoming 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

### 6050Outgoing 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.

### 6051Description

60521. This command allows saving a loaded key outside the TPM. After creation of the  
6053 keyContextBlob, the TPM automatically releases the internal memory used by that key.  
6054 The format of the key context blob is specific to a TPM.

60552. A TPM protected capability belonging to the TPM that created a key context blob MUST  
6056 be the only entity that can interpret the contents of that blob. If a cryptographic  
6057 technique is used for this purpose, the level of security provided by that technique  
6058 SHALL be at least as secure as a 2048 bit RSA algorithm. Any secrets (such as keys)

6059 used in such a cryptographic technique MUST be generated using the TPM's random  
6060 number generator. Any symmetric key MUST be used within the power-on session  
6061 during which it was created, only.

60623. A key context blob SHALL enable verification of the integrity of the contents of the blob  
6063 by a TPM protected capability.

60644. A key context blob SHALL enable verification of the session validity of the contents of the  
6065 blob by a TPM protected capability. The method SHALL ensure that all key context blobs  
6066 are rendered invalid if power to the TPM is interrupted.

## 6068 27.2.2 TPM\_LoadKeyContext

### 6069 Start of informative comment:

6070 TPM\_LoadKeyContext loads a key context blob into the TPM previously retrieved by a  
6071 TPM\_SaveKeyContext call. After successful completion the handle returned by this  
6072 command can be used to access the key.

### 6073 End of informative comment.

### 6074 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.

### 6075 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.

### 6076 Description

6077 1. This command allows loading a key context blob into the TPM previously retrieved by a  
6078 TPM\_SaveKeyContext call. After successful completion the handle returned by this  
6079 command can be used to access the key.

6080 2. The contents of a key context blob SHALL be discarded unless the contents have passed  
6081 an integrity test. This test SHALL (statistically) prove that the contents of the blob are  
6082 the same as when the blob was created.

6083 3. The contents of a key context blob SHALL be discarded unless the contents have passed  
6084 a session validity test. This test SHALL (statistically) prove that the blob was created by  
6085 this TPM during this power-on session.

6086

## 608727.2.3 TPM\_SaveAuthContext

### 6088Start of informative comment:

6089TPM\_SaveAuthContext saves a loaded authorization session outside the TPM. After creation  
6090of the authorization context blob, the TPM automatically releases the internal memory used  
6091by that session. The format of the authorization context blob is specific to a TPM.

### 6092End of informative comment.

### 6093Incoming 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

### 6094Outgoing 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.

### 6095Description

6096This command allows saving a loaded authorization session outside the TPM. After creation  
6097of the authContextBlob, the TPM automatically releases the internal memory used by that  
6098session. The format of the authorization context blob is specific to a TPM.

6099A TPM protected capability belonging to the TPM that created an authorization context blob  
6100MUST be the only entity that can interpret the contents of that blob. If a cryptographic  
6101technique is used for this purpose, the level of security provided by that technique SHALL  
6102be at least as secure as a 2048 bit RSA algorithm. Any secrets (such as keys) used in such a  
6103cryptographic technique MUST be generated using the TPM's random number generator.  
6104Any symmetric key MUST be used within the power-on session during which it was created,  
6105only.

6106An authorization context blob SHALL enable verification of the integrity of the contents of  
6107the blob by a TPM protected capability.



6108An authorization context blob SHALL enable verification of the session validity of the  
6109contents of the blob by a TPM protected capability. The method SHALL ensure that all  
6110authorization context blobs are rendered invalid if power to the TPM is interrupted.

## 6111**27.2.4 TPM\_LoadAuthContext**

### 6112**Start of informative comment:**

6113TPM\_LoadAuthContext loads an authorization context blob into the TPM previously  
6114retrieved by a TPM\_SaveAuthContext call. After successful completion, the handle returned  
6115by this command can be used to access the authorization session.

### 6116**End of informative comment.**

### 6117**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.

### 6118**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.

### 6119**Description**

6120This command allows loading an authorization context blob into the TPM previously  
6121retrieved by a TPM\_SaveAuthContext call. After successful completion, the handle returned  
6122by this command can be used to access the authorization session.

6123The contents of an authorization context blob SHALL be discarded unless the contents have  
6124passed an integrity test. This test SHALL (statistically) prove that the contents of the blob  
6125are the same as when the blob was created.

6126The contents of an authorization context blob SHALL be discarded unless the contents have  
6127passed a session validity test. This test SHALL (statistically) prove that the blob was created  
6128by this TPM during this power-on session.

6129For an OSAP authorization context blob referring to a key, verify that the key linked to this  
6130session is resident in the TPM.

6131

## 6132 **27.3 DIR commands**

### 6133 **Start of informative comment:**

6134 The DIR commands are replaced by the NV storage commands.

6135 The DIR [0] in 1.1 is now TPM\_PERMANENT\_DATA -> authDIR[0] and is always available for  
6136 the TPM to use. It is accessed by DIR commands using dirIndex 0 and by NV commands  
6137 using nvIndex TPM\_NV\_INDEX\_DIR.

6138 If the TPM vendor supports additional DIR registers, the TPM vendor may return errors or  
6139 provide vendor specific mappings for those DIR registers to NV storage locations.

### 6140 **End of informative comment.**

6141 1. A dirIndex value of 0 MUST corresponds to an NV storage nvIndex value  
6142 TPM\_NV\_INDEX\_DIR.

6143 2. The TPM vendor MAY return errors or MAY provide vendor specific mappings for DIR  
6144 dirIndex values greater than 0 to NV storage locations.

## 6145**27.3.1 TPM\_DirWriteAuth**

### 6146**Start of informative comment:**

6147The TPM\_DirWriteAuth operation provides write access to the Data Integrity Registers. DIRs  
6148are non-volatile memory registers held in a TPM-shielded location. Owner authentication is  
6149required to authorize this action.

6150Access is also provided through the NV commands with nvIndex TPM\_NV\_INDEX\_DIR.  
6151Owner authorization is not required when nvLocked is FALSE.

6152Version 1.2 requires only one DIR. If the DIR named does not exist, the TPM\_DirWriteAuth  
6153operation returns TPM\_BADINDEX.

### 6154**End of informative comment.**

### 6155**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.

### 6156**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.

### 6157**Actions**

61581. Validate that authHandle contains a TPM Owner AuthData to execute the  
6159 TPM\_DirWriteAuth command
61602. Validate that dirIndex points to a valid DIR on this TPM
61613. Write newContents into the DIR pointed to by dirIndex

6162

## 616327.3.2 TPM\_DirRead

### 6164Start of informative comment:

6165The TPM\_DirRead operation provides read access to the DIRs. No authentication is required  
6166to perform this action because typically no cryptographically useful AuthData is available  
6167early in boot. TSS implementers may choose to provide other means of authorizing this  
6168action. Version 1.2 requires only one DIR. If the DIR named does not exist, the  
6169TPM\_DirRead operation returns TPM\_BADINDEX.

### 6170End of informative comment.

### 6171Incoming 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

### 6172Outgoing 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

### 6173Actions

61741. Validate that dirIndex points to a valid DIR on this TPM

61752. Return the contents of the DIR in dirContents

## 6176 **27.4 Change Auth**

6177 **Start of informative comment:**

6178 The change auth commands can be duplicated by creating a transport session with  
6179 confidentiality and issuing the changeAuth command.

6180 **End of informative comment.**

6181

## 6182**27.4.1 TPM\_ChangeAuthAsymStart**

### 6183**Start of informative comment:**

6184The TPM\_ChangeAuthAsymStart starts the process of changing AuthData for an entity. It  
6185sets up an OIAP session that must be retained for use by its twin  
6186TPM\_ChangeAuthAsymFinish command.

6187TPM\_ChangeAuthAsymStart creates a temporary asymmetric public key “tempkey” to  
6188provide confidentiality for new AuthData to be sent to the TPM. TPM\_ChangeAuthAsymStart  
6189certifies that tempkey was generated by a genuine TPM, by generating a certifyInfo  
6190structure that is signed by a TPM identity. The owner of that TPM identity must cooperate  
6191to produce this command, because TPM\_ChangeAuthAsymStart requires authorization to  
6192use that identity.

6193It is envisaged that tempkey and certifyInfo are given to the owner of the entity whose  
6194authorization is to be changed. That owner uses certifyInfo and a  
6195TPM\_IDENTITY\_CREDENTIAL to verify that tempkey was generated by a genuine TPM. This  
6196is done by verifying the TPM\_IDENTITY\_CREDENTIAL using the public key of a CA,  
6197verifying the signature on the certifyInfo structure with the public key of the identity in  
6198TPM\_IDENTITY\_CREDENTIAL, and verifying tempkey by comparing its digest with the value  
6199inside certifyInfo. The owner uses tempkey to encrypt the desired new AuthData and inserts  
6200that encrypted data in a TPM\_ChangeAuthAsymFinish command, in the knowledge that  
6201only a TPM with a specific identity can interpret the new AuthData.

### 6202**End of informative comment.**

### 6203**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.



## 6204 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.

## 6205 Actions

62061. The TPM SHALL verify the AuthData to use the TPM identity key held in idHandle. The  
6207 TPM MUST verify that the key is a TPM identity key.
62082. The TPM SHALL validate the algorithm parameters for the key to create from the  
6209 tempKey parameter.
62103. Recommended key type is RSA
62114. Minimum RSA key size MUST is 512 bits, recommended RSA key size is 1024
62125. For other key types the minimum key size strength MUST be comparable to RSA 512
62136. If the TPM is not designed to create a key of the requested type, return the error code  
6214 TPM\_BAD\_KEY\_PROPERTY
62157. The TPM SHALL create a new key (k1) in accordance with the algorithm parameter. The  
6216 newly created key is pointed to by ephHandle.
62178. The TPM SHALL fill in all fields in tempKey using k1 for the information. The TPM\_KEY  
6218 -> encSize MUST be 0.
62199. The TPM SHALL fill in certifyInfo using k1 for the information. The certifyInfo -> data  
6220 field is supplied by the antiReplay.
622110. The TPM then signs the certifyInfo parameter using the key pointed to by idHandle. The  
6222 resulting signed blob is returned in sig parameter

## 6223Field Descriptions for certifyInfo parameter

Type	Name	Description
TPM_VERSION	Version	TPM version structure; Part 2 TPM_VERSION
keyFlags	Redirection	This SHALL be set to FALSE
	Migratable	This SHALL be set to FALSE
	Volatile	This SHALL be set to TRUE
TPM_AUTH_DATA_USAGE	authDataUsage	This SHALL be set to TPM_AUTH_NEVER
TPM_KEY_USAGE	KeyUsage	This SHALL be set to TPM_KEY_AUTHCHANGE
UINT32	PCRInfoSize	This SHALL be set to 0
TPM_DIGEST	pubDigest	This SHALL be the hash of the public key being certified.
TPM_NONCE	Data	This SHALL be set to antiReplay
TPM_KEY_PARMS	info	This specifies the type of key and its parameters.
BOOL	parentPCRStatus	This SHALL be set to FALSE.

## 6225**27.4.2 TPM\_ChangeAuthAsymFinish**

### 6226**Start of informative comment:**

6227The TPM\_ChangeAuthAsymFinish command allows the owner of an entity to change the  
6228AuthData for the entity.

6229The command requires the cooperation of the owner of the parent of the entity, since  
6230AuthData must be provided to use that parent entity. The command requires knowledge of  
6231the existing AuthData information and passes the new AuthData information. The  
6232newAuthLink parameter proves knowledge of existing AuthData information and new  
6233AuthData information. The new AuthData information “encNewAuth” is encrypted using the  
6234“tempKey” variable obtained via TPM\_ChangeAuthAsymStart.

6235A parent therefore retains control over a change in the AuthData of a child, but is prevented  
6236from knowing the new AuthData for that child.

6237The changeProof parameter provides a proof that the new AuthData value was properly  
6238inserted into the entity. The inclusion of a nonce from the TPM provides an entropy source  
6239in the case where the AuthData value may be in itself be a low entropy value (hash of a  
6240password etc).

### 6241**End of informative comment.**

### 6242**Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuthAsymFinish
4	4			TPM_KEY_HANDLE	parentHandle	The keyHandle of the parent key for the input data
5	4			TPM_KEY_HANDLE	ephHandle	The keyHandle identifier for the ephemeral key
6	2	3S	2	TPM_ENTITY_TYPE	entityType	The type of entity to be modified
7	20	4s	20	TPM_HMAC	newAuthLink	HMAC calculation that links the old and new AuthData values together
8	4	5S	4	UINT32	newAuthSize	Size of encNewAuth
9	<>	6S	<>	BYTE[]	encNewAuth	New AuthData encrypted with ephemeral key.
10	4	7S	4	UINT32	encDataSize	The size of the inData parameter
11	<>	8S	<>	BYTE[]	encData	The encrypted entity that is to be modified.
12	4			TPM_AUTHHANDLE	authHandle	Authorization for parent key.
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
13	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
14	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
15	20			TPM_AUTHDATA	privAuth	The authorization session digest for inputs and parentHandle. HMAC key: parentKey.usageAuth.

6243

## 6244Outgoing Operands and Sizes

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RSP_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of output bytes including paramSize and tag
3	4	1S	4	TPM_RESULT	returnCode	The return code of the operation.
		2S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_ChangeAuthAsymFinish
4	4	3S	4	UINT32	outDataSize	The used size of the output area for outData
5	<>	4S	<>	BYTE[]	outData	The modified, encrypted entity.
6	20	5s	20	TPM_NONCE	saltNonce	A nonce value from the TPM RNG to add entropy to the changeProof value
7	<>	6S	<>	TPM_DIGEST	changeProof	Proof that AuthData has changed.
8	20	2H1	20	TPM_NONCE	nonceEven	Even nonce newly generated by TPM to cover outputs
		3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
9	1	4H1	1	BOOL	continueAuthSession	Continue use flag, TRUE if handle is still active
10	20			TPM_AUTHDATA	resAuth	The authorization session digest for the returned parameters. HMAC key: parentKey.usageAuth.

## 6245Description

6246If the parentHandle points to the SRK then the HMAC key MUST be built using the TPM  
6247Owner authentication.

## 6248Actions

62491. The TPM SHALL validate that the authHandle parameter authorizes use of the key in  
6250 parentHandle.

62512. The encData field MUST be the encData field from TPM\_STORED\_DATA or TPM\_KEY.

62523. The TPM SHALL create e1 by decrypting the entity held in the encData parameter.

62534. The TPM SHALL create a1 by decrypting encNewAuth using the ephHandle ->  
6254 TPM\_KEY\_AUTHCHANGE private key. a1 is a structure of type  
6255 TPM\_CHANGEAUTH\_VALIDATE.

62565. The TPM SHALL create b1 by performing the following HMAC calculation: b1 = HMAC  
6257 (a1 -> newAuthSecret). The secret for this calculation is encData -> currentAuth. This  
6258 means that b1 is a value built from the current AuthData value (encData ->  
6259 currentAuth) and the new AuthData value (a1 -> newAuthSecret).

62606. The TPM SHALL compare b1 with newAuthLink. The TPM SHALL indicate a failure if the  
6261 values do not match.

62627. The TPM SHALL replace e1 -> authData with a1 -> newAuthSecret

62638. The TPM SHALL encrypt e1 using the appropriate functions for the entity type. The key  
6264 to encrypt with is parentHandle.

62659. The TPM SHALL create saltNonce by taking the next 20 bytes from the TPM RNG.

6266 10. The TPM SHALL create changeProof a HMAC of (saltNonce concatenated with a1 -> n1)  
6267 using a1 -> newAuthSecret as the HMAC secret.

6268 11. The TPM MUST destroy the TPM\_KEY\_AUTHCHANGE key associated with the  
6269 authorization session.

## 6270**27.5 TPM\_Reset**

### 6271**Start of informative comment:**

6272TPM\_Reset releases all resources associated with existing authorization sessions. This is  
6273useful if a TSS driver has lost track of the state in the TPM.

### 6274**End of informative comment.**

6275Deprecated Command in 1.2

### 6276**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.

### 6277**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.

### 6278**Description**

6279This is a deprecated command in V1.2. This command in 1.1 only referenced authorization  
6280sessions and is not upgraded to affect any other TPM entity in 1.2

### 6281**Actions**

62821. The TPM invalidates all resources allocated to authorization sessions as per version 1.1  
6283 extant in the TPM

6284 a. This includes structures created by TPM\_SaveAuthContext and TPM\_SaveKeyContext

6285 b. The TPM MUST invalidate OSAP sessions

6286 c. The TPM MAY invalidate DSAP sessions

6287 d. The TPM MUST NOT invalidate structures created by TPM\_SaveContext

62882. The TPM does not reset any PCR or DIR values.

62893. The TPM does not reset any flags in the TPM\_STCLEAR\_FLAGS structure.

62904. The TPM does not reset or invalidate any keys

## 629127.6 TPM\_OwnerReadPubek

### 6292Start of informative comment:

6293Return the endorsement key public portion. This is authorized by the TPM Owner.

### 6294End of informative comment.

### 6295Incoming 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.

### 6296Outgoing 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.

### 6297Description

6298This command returns the PUBEK.

### 6299Actions

6300The TPM\_OwnerReadPubek command SHALL

63011. Validate the TPM Owner AuthData to execute this command

63022. Export the PUBEK

## 630327.7 TPM\_DisablePubekRead

### 6304Start of informative comment:

6305The TPM Owner may wish to prevent any entity from reading the PUBEK. This command  
6306sets the non-volatile flag so that the TPM\_ReadPubek command always returns  
6307TPM\_DISABLED\_CMD.

6308This command has in essence been deprecated as TPM\_TakeOwnership now sets the value  
6309to false. The command remains at this time for backward compatibility.

### 6310End of informative comment.

### 6311Incoming 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.

### 6312Outgoing 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.

### 6313Actions

63141. This capability sets the TPM\_PERMANENT\_FLAGS -> readPubek flag to FALSE.



## 6315 **27.8 TPM\_LoadKey**

### 6316 **Start of informative comment:**

6317 Version 1.2 deprecates TPM\_LoadKey due to the HMAC of the new key handle on return.  
6318 The wrapping makes use of the handle difficult in an environment where the TSS, or other  
6319 management entity, is changing the TPM handle to a virtual handle.

6320 Software using TPM\_LoadKey on a 1.2 TPM can have a collision with the returned handle as  
6321 the 1.2 TPM uses random values in the lower three bytes of the handle. All new software  
6322 must use LoadKey2 to allow management software the ability to manage the key handle.

6323 Before the TPM can use a key to either wrap, unwrap, bind, unbind, seal, unseal, sign or  
6324 perform any other action, it needs to be present in the TPM. The TPM\_LoadKey function  
6325 loads the key into the TPM for further use.

6326 The TPM assigns the key handle. The TPM always locates a loaded key by use of the handle.  
6327 The assumption is that the handle may change due to key management operations. It is the  
6328 responsibility of upper level software to maintain the mapping between handle and any  
6329 label used by external software.

6330 This command has the responsibility of enforcing restrictions on the use of keys. For  
6331 example, when attempting to load a STORAGE key it will be checked for the restrictions on  
6332 a storage key (2048 size etc.).

6333 The load command must maintain a record of whether any previous key in the key  
6334 hierarchy was bound to a PCR using parentPCRStatus.

6335 The flag parentPCRStatus enables the possibility of checking that a platform passed  
6336 through some particular state or states before finishing in the current state. A grandparent  
6337 key could be linked to state-1, a parent key could be linked to state-2, and a child key could be  
6338 linked to state-3, for example. The use of the child key then indicates that the platform  
6339 passed through states 1 and 2 and is currently in state 3, in this example. TPM\_Startup  
6340 with stType == TPM\_ST\_CLEAR indicates that the platform has been reset, so the platform  
6341 has not passed through the previous states. Hence keys with parentPCRStatus==TRUE  
6342 must be unloaded if TPM\_Startup is issued with stType == TPM\_ST\_CLEAR.

6343 If a TPM\_KEY structure has been decrypted AND the integrity test using "pubDataDigest"  
6344 has passed AND the key is non-migratory, the key must have been created by the TPM. So  
6345 there is every reason to believe that the key poses no security threat to the TPM. While there  
6346 is no known attack from a rogue migratory key, there is a desire to verify that a loaded  
6347 migratory key is a real key, arising from a general sense of unease about execution of  
6348 arbitrary data as a key. Ideally a consistency check would consist of an encrypt/decrypt  
6349 cycle, but this may be expensive. For RSA keys, it is therefore suggested that the  
6350 consistency test consists of dividing the supposed RSA product by the supposed RSA prime,  
6351 and checking that there is no remainder.

### 6352 **End of informative comment.**

### 6353Incoming Operands and Sizes

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

### 6354Outgoing Operands and Sizes

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

### 6355Actions

6356The TPM SHALL perform the following steps:

63571. Validate the command and the parameters using parentAuth and parentHandle ->  
6358 usageAuth

63592. If parentHandle -> keyUsage is NOT TPM\_KEY\_STORAGE return  
6360 TPM\_INVALID\_KEYUSAGE

63613. If the TPM is not designed to operate on a key of the type specified by inKey, return the  
6362 error code TPM\_BAD\_KEY\_PROPERTY

63634. The TPM MUST handle both TPM\_KEY and TPM\_KEY12 structures

63645. Decrypt the inKey -> privkey to obtain TPM\_STORE\_ASYMKEY structure using the key  
6365 in parentHandle

63666. Validate the integrity of inKey and decrypted TPM\_STORE\_ASYMKEY
- 6367 a. Reproduce inKey -> TPM\_STORE\_ASYMKEY -> pubDataDigest using the fields of  
6368 inKey, and check that the reproduced value is the same as pubDataDigest
63697. Validate the consistency of the key and its key usage.
- 6370 a. If inKey -> keyFlags -> migratable is TRUE, the TPM SHALL verify consistency of the  
6371 public and private components of the asymmetric key pair. If inKey -> keyFlags ->  
6372 migratable is FALSE, the TPM MAY verify consistency of the public and private  
6373 components of the asymmetric key pair. The consistency of an RSA key pair MAY be  
6374 verified by dividing the supposed (P\*Q) product by a supposed prime and checking that  
6375 there is no remainder.
- 6376 b. If inKey -> keyUsage is TPM\_KEY\_IDENTITY, verify that inKey->keyFlags->migratable  
6377 is FALSE. If it is not, return TPM\_INVALID\_KEYUSAGE
- 6378 c. If inKey -> keyUsage is TPM\_KEY\_AUTHCHANGE, return TPM\_INVALID\_KEYUSAGE
- 6379 d. If inKey -> keyFlags -> migratable equals 0 then verify that TPM\_STORE\_ASYMKEY  
6380 -> migrationAuth equals TPM\_PERMANENT\_DATA -> tpmProof
- 6381 e. Validate the mix of encryption and signature schemes
- 6382 f. If TPM\_PERMANENT\_FLAGS -> FIPS is TRUE then
- 6383 i. If keyInfo -> keySize is less than 1024 return TPM\_NOTFIPS
- 6384 ii. If keyInfo -> authDataUsage specifies TPM\_AUTH\_NEVER return  
6385 TPM\_NOTFIPS
- 6386 iii. If keyInfo -> keyUsage specifies TPM\_KEY\_LEGACY return TPM\_NOTFIPS
- 6387 g. If inKey -> keyUsage is TPM\_KEY\_STORAGE or TPM\_KEY\_MIGRATE
- 6388 i. algorithmID MUST be TPM\_ALG\_RSA
- 6389 ii. Key size MUST be 2048
- 6390 iii. exponentSize MUST be 0
- 6391 iv. sigScheme MUST be TPM\_SS\_NONE
- 6392 h. If inKey -> keyUsage is TPM\_KEY\_IDENTITY
- 6393 i. algorithmID MUST be TPM\_ALG\_RSA
- 6394 ii. Key size MUST be 2048
- 6395 iii. exponentSize MUST be 0
- 6396 iv. encScheme MUST be TPM\_ES\_NONE
- 6397 i. If the decrypted inKey -> pcrInfo is NULL,
- 6398 i. The TPM MUST set the internal indicator to indicate that the key is not using  
6399 any PCR registers.
- 6400 j. Else
- 6401 i. The TPM MUST store pcrInfo in a manner that allows the TPM to calculate a  
6402 composite hash whenever the key will be in use

6403           ii. The TPM MUST handle both version 1.1 TPM\_PCR\_INFO and 1.2  
6404           TPM\_PCR\_INFO\_LONG structures according to the type of TPM\_KEY structure  
6405           (1) The TPM MUST validate the TPM\_PCR\_INFO or TPM\_PCR\_INFO\_LONG  
6406           structures for legal values. However, the digestAtRelease and  
6407           localityAtRelease are not validated for authorization until use time.

64088. Perform any processing necessary to make TPM\_STORE\_ASYMKEY key available for  
6409 operations

64109. Load key and key information into internal memory of the TPM. If insufficient memory  
6411 exists, return error TPM\_NOSPACE.

641210. Assign inKeyHandle according to internal TPM rules.

641311. Set InKeyHandle -> parentPCRStatus to parentHandle -> parentPCRStatus.

641412. If parentHandle indicates it is using PCR registers, then set inKeyHandle ->  
6415 parentPCRStatus to TRUE.

## 6416**28. Deleted Commands**

6417**Start of informative comment:**

6418These commands are no longer active commands. Their removal is due to security concerns  
6419with their use.

6420**End of informative comment.**

64211. The TPM MUST return TPM\_BAD\_ORDINAL for any deleted command

## 6422**28.1 TPM\_GetCapabilitySigned**

### 6423**Start of informative comment:**

6424Along with TPM\_GetCapabilityOwner this command allowed the possible signature of  
6425improper values.

6426TPM\_GetCapabilitySigned is almost the same as TPM\_GetCapability. The differences are  
6427that the input includes a challenge (a nonce) and the response includes a digital signature  
6428to vouch for the source of the answer.

6429If a caller itself requires proof, it is sufficient to use any signing key for which only the TPM  
6430and the caller have AuthData.

6431If a caller requires proof for a third party, the signing key must be one whose signature is  
6432trusted by the third party. A TPM-identity key may be suitable.

### 6433**End of informative comment.**

### 6434**Deleted Ordinal**

6435TPM\_GetCapabilitySigned

## 643628.2 TPM\_GetOrdinalAuditStatus

6437**Start of informative comment:**

6438Get the status of the audit flag for the given ordinal.

6439**End of informative comment.**

### 6440Incoming 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

### 6441Outgoing 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

### 6442Actions

64431. The TPM returns the Boolean value for the given ordinal. The value is TRUE if the  
6444 command is being audited.

## 6445**28.3 TPM\_CertifySelfTest**

### 6446**Start of informative comment:**

6447TPM\_CertifySelfTest causes the TPM to perform a full self-test and return an authenticated  
6448value if the test passes.

6449If a caller itself requires proof, it is sufficient to use any signing key for which only the TPM  
6450and the caller have AuthData.

6451If a caller requires proof for a third party, the signing key must be one whose signature is  
6452trusted by the third party. A TPM-identity key may be suitable.

### 6453**End of informative comment.**

### 6454**Incoming Operands and Sizes**

PARAM		HMAC		Type	Name	Description
#	SZ	#	SZ			
1	2			TPM_TAG	tag	TPM_TAG_RQU_AUTH1_COMMAND
2	4			UINT32	paramSize	Total number of input bytes including paramSize and tag
3	4	1S	4	TPM_COMMAND_CODE	ordinal	Command ordinal: TPM_ORD_CertifySelfTest
4	4			TPM_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can perform digital signatures.
5	20	2S	20	TPM_NONCE	antiReplay	Anti Replay nonce to prevent replay of messages
6	4			TPM_AUTHHANDLE	authHandle	The authorization session handle used for keyHandle authorization
		2H1	20	TPM_NONCE	authLastNonceEven	Even nonce previously generated by TPM to cover inputs
7	20	3H1	20	TPM_NONCE	nonceOdd	Nonce generated by system associated with authHandle
8	1	4H1	1	BOOL	continueAuthSession	The continue use flag for the authorization session handle
9	20			TPM_AUTHDATA	privAuth	The authorization session digest that authorizes the inputs and use of keyHandle. HMAC key: key.usageAuth

### 6455**Outgoing Operands and Sizes**

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



**6456 Description**

6457 The key in keyHandle MUST have a KEYUSAGE value of type TPM\_KEY\_SIGNING or  
6458 TPM\_KEY\_LEGACY or TPM\_KEY\_IDENTITY.

6459 Information returned by TPM\_CertifySelfTest MUST NOT aid identification of an individual  
6460 TPM.

**6461 Actions**

6462 1. The TPM SHALL perform TPM\_SelfTestFull. If the test fails the TPM returns the  
6463 appropriate error code.

6464 2. After successful completion of the self-test the TPM then validates the authorization to  
6465 use the key pointed to by keyHandle

6466 a. If the key pointed to by keyHandle has a signature scheme that is not  
6467 TPM\_SS\_RSASSAPKCS1v15\_SHA1, the TPM may either return TPM\_BAD\_SCHEME or  
6468 may return TPM\_SUCCESS and a vendor specific signature.

6469 3. Create t1 the NOT null terminated string of "Test Passed", i.e. 11 bytes.

6470 4. The TPM creates m2 the message to sign by concatenating t1 || AntiReplay || ordinal.

6471 5. The TPM signs the SHA-1 of m2 using the key identified by keyHandle, and returns the  
6472 signature as sig.

6473

## 6474**28.4 TPM\_GetAuditEvent**

6475

6476**Start of informative comment:**

6477Deleted

6478**End of informative comment.**

6479**Deleted Ordinal**

6480TPM\_GetAuditEvent

## 6481 **28.5 TPM\_GetAuditEventSigned**

6482

6483 **Start of informative comment:**

6484 Deleted

6485 **End of informative comment.**

6486 **Deleted Ordinal**

6487 TPM\_GetAuditEventSigned

6488

6489