

# Trusted Platform Module Library

## Part 3: Commands

Family “2.0”

Level 00 Revision 00.96

March 15, 2013

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

## Published

Copyright © TCG 2006-2013

## Licenses and Notices

### 1. Copyright Licenses:

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

### 2. Source Code Distribution Conditions:

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

### 3. Disclaimers:

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

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

## CONTENTS

1	Scope .....	1
2	Terms and Definitions .....	1
3	Symbols and abbreviated terms.....	1
4	Notation .....	1
4.1	Introduction .....	1
4.2	Table Decorations.....	1
4.3	Handle and Parameter Demarcation .....	3
4.4	AuthorizationSize and ParameterSize .....	3
5	Normative References.....	4
6	Symbols and Abbreviated Terms .....	4
7	Command Processing .....	4
7.1	Introduction .....	4
7.2	Command Header Validation.....	4
7.3	Mode Checks .....	4
7.4	Handle Area Validation .....	5
7.5	Session Area Validation.....	6
7.6	Authorization Checks.....	7
7.7	Parameter Decryption.....	8
7.8	Parameter Unmarshaling.....	8
7.9	Command Post Processing .....	10
8	Response Values .....	12
8.1	Tag.....	12
8.2	Response Codes .....	12
9	Implementation Dependent .....	15
10	Detailed Actions Assumptions.....	16
10.1	Introduction .....	16
10.2	Pre-processing.....	16
10.3	Post Processing.....	16
11	Start-up.....	17
11.1	Introduction .....	17
11.2	_TPM_Init.....	17
11.3	TPM2_Startup.....	19
11.4	TPM2_Shutdown .....	24
12	Testing.....	28
12.1	Introduction .....	28
12.2	TPM2_SelfTest .....	29
12.3	TPM2_IncrementalSelfTest .....	32
12.4	TPM2_GetTestResult .....	35
13	Session Commands .....	38
13.1	TPM2_StartAuthSession .....	38
13.2	TPM2_PolicyRestart .....	43
14	Object Commands.....	46

14.1	TPM2_Create.....	46
14.2	TPM2_Load .....	51
14.3	TPM2_LoadExternal .....	55
14.4	TPM2_ReadPublic.....	60
14.5	TPM2_ActivateCredential .....	63
14.6	TPM2_MakeCredential .....	67
14.7	TPM2_Unseal .....	70
14.8	TPM2_ObjectChangeAuth.....	73
15	Duplication Commands .....	77
15.1	TPM2_Duplicate .....	77
15.2	TPM2_Rewrap.....	81
15.3	TPM2_Import.....	86
16	Asymmetric Primitives .....	91
16.1	Introduction .....	91
16.2	TPM2_RSA_Encrypt.....	91
16.3	TPM2_RSA_Decrypt .....	96
16.4	TPM2_ECDH_KeyGen .....	100
16.5	TPM2_ECDH_ZGen .....	103
16.6	TPM2_ECC_Parameters .....	106
16.1	TPM2_ZGen_2Phase .....	108
17	Symmetric Primitives.....	112
17.1	Introduction .....	112
17.2	TPM2_EncryptDecrypt.....	114
17.3	TPM2_Hash.....	118
17.4	TPM2_HMAC.....	121
18	Random Number Generator.....	125
18.1	TPM2_GetRandom.....	125
18.2	TPM2_StirRandom .....	128
19	Hash/HMAC/Event Sequences .....	131
19.1	Introduction .....	131
19.2	TPM2_HMAC_Start .....	131
19.3	TPM2_HashSequenceStart .....	135
19.4	TPM2_SequenceUpdate .....	138
19.5	TPM2_SequenceComplete.....	142
19.6	TPM2_EventSequenceComplete .....	146
20	Attestation Commands .....	150
20.1	Introduction .....	150
20.2	TPM2_Certify.....	152
20.3	TPM2_CertifyCreation .....	156
20.4	TPM2_Quote.....	160
20.5	TPM2_GetSessionAuditDigest .....	164
20.6	TPM2_GetCommandAuditDigest .....	168
20.7	TPM2_GetTime.....	172
21	Ephemeral EC Keys .....	176

21.1	Introduction .....	176
21.2	TPM2_Commit .....	177
21.3	TPM2_EC_Ephemeral .....	183
22	Signing and Signature Verification .....	186
22.1	TPM2_VerifySignature .....	186
22.2	TPM2_Sign .....	190
23	Command Audit .....	194
23.1	Introduction .....	194
23.2	TPM2_SetCommandCodeAuditStatus .....	195
24	Integrity Collection (PCR) .....	199
24.1	Introduction .....	199
24.2	TPM2_PCR_Extend .....	200
24.3	TPM2_PCR_Event .....	203
24.4	TPM2_PCR_Read .....	206
24.5	TPM2_PCR_Allocate .....	209
24.6	TPM2_PCR_SetAuthPolicy .....	212
24.7	TPM2_PCR_SetAuthValue .....	215
24.8	TPM2_PCR_Reset .....	218
24.9	_TPM_Hash_Start .....	221
24.10	_TPM_Hash_Data .....	223
24.11	_TPM_Hash_End .....	225
25	Enhanced Authorization (EA) Commands .....	228
25.1	Introduction .....	228
25.2	Signed Authorization Actions .....	229
25.3	TPM2_PolicySigned .....	233
25.4	TPM2_PolicySecret .....	239
25.5	TPM2_PolicyTicket .....	243
25.6	TPM2_PolicyOR .....	247
25.7	TPM2_PolicyPCR .....	250
25.8	TPM2_PolicyLocality .....	254
25.9	TPM2_PolicyNV .....	258
25.10	TPM2_PolicyCounterTimer .....	263
25.11	TPM2_PolicyCommandCode .....	268
25.12	TPM2_PolicyPhysicalPresence .....	271
25.13	TPM2_PolicyCpHash .....	274
25.14	TPM2_PolicyNameHash .....	278
25.15	TPM2_PolicyDuplicationSelect .....	281
25.16	TPM2_PolicyAuthorize .....	285
25.17	TPM2_PolicyAuthValue .....	289
25.18	TPM2_PolicyPassword .....	292
25.19	TPM2_PolicyGetDigest .....	295
26	Hierarchy Commands .....	298
26.1	TPM2_CreatePrimary .....	298
26.2	TPM2_HierarchyControl .....	302
26.3	TPM2_SetPrimaryPolicy .....	306
26.4	TPM2_ChangePPS .....	309

26.5	TPM2_ChangeEPS .....	312
26.6	TPM2_Clear.....	315
26.7	TPM2_ClearControl .....	319
26.8	TPM2_HierarchyChangeAuth.....	322
27	Dictionary Attack Functions.....	325
27.1	Introduction .....	325
27.2	TPM2_DictionaryAttackLockReset .....	325
27.3	TPM2_DictionaryAttackParameters.....	328
28	Miscellaneous Management Functions.....	331
28.1	Introduction .....	331
28.2	TPM2_PP_Commands .....	331
28.3	TPM2_SetAlgorithmSet .....	334
29	Field Upgrade.....	337
29.1	Introduction .....	337
29.2	TPM2_FieldUpgradeStart.....	339
29.3	TPM2_FieldUpgradeData .....	342
29.4	TPM2_FirmwareRead.....	345
30	Context Management.....	348
30.1	Introduction .....	348
30.2	TPM2_ContextSave.....	348
30.3	TPM2_ContextLoad .....	353
30.4	TPM2_FlushContext .....	358
30.5	TPM2_EvictControl.....	361
31	Clocks and Timers.....	366
31.1	TPM2_ReadClock.....	366
31.2	TPM2_ClockSet.....	369
31.3	TPM2_ClockRateAdjust.....	372
32	Capability Commands .....	375
32.1	Introduction .....	375
32.2	TPM2_GetCapability.....	375
32.3	TPM2_TestParms .....	383
33	Non-volatile Storage.....	386
33.1	Introduction .....	386
33.2	NV Counters .....	387
33.3	TPM2_NV_DefineSpace.....	388
33.4	TPM2_NV_UndefineSpace.....	394
33.5	TPM2_NV_UndefineSpaceSpecial.....	397
33.6	TPM2_NV_ReadPublic.....	400
33.7	TPM2_NV_Write .....	403
33.8	TPM2_NV_Increment .....	407
33.9	TPM2_NV_Extend .....	411
33.10	TPM2_NV_SetBits .....	415
33.11	TPM2_NV_WriteLock .....	419
33.12	TPM2_NV_GlobalWriteLock.....	423

33.13 TPM2_NV_Read.....	426
33.14 TPM2_NV_ReadLock.....	429
33.15 TPM2_NV_ChangeAuth.....	432
33.16 TPM2_NV_Certify.....	435

**Tables**

Table 1 — Command Modifiers and Decoration .....	2
Table 2 — Separators .....	3
Table 3 — Unmarshaling Errors .....	10
Table 4 — Command-Independent Response Codes .....	13
Table 5 — TPM2_Startup Command .....	21
Table 6 — TPM2_Startup Response .....	21
Table 7 — TPM2_Shutdown Command .....	25
Table 8 — TPM2_Shutdown Response .....	25
Table 9 — TPM2_SelfTest Command .....	30
Table 10 — TPM2_SelfTest Response .....	30
Table 11 — TPM2_IncrementalSelfTest Command .....	33
Table 12 — TPM2_IncrementalSelfTest Response .....	33
Table 13 — TPM2_GetTestResult Command .....	36
Table 14 — TPM2_GetTestResult Response .....	36
Table 15 — TPM2_StartAuthSession Command .....	40
Table 16 — TPM2_StartAuthSession Response .....	40
Table 17 — TPM2_PolicyRestart Command .....	44
Table 18 — TPM2_PolicyRestart Response .....	44
Table 19 — TPM2_Create Command .....	48
Table 20 — TPM2_Create Response .....	48
Table 21 — TPM2_Load Command .....	52
Table 22 — TPM2_Load Response .....	52
Table 23 — TPM2_LoadExternal Command .....	57
Table 24 — TPM2_LoadExternal Response .....	57
Table 25 — TPM2_ReadPublic Command .....	61
Table 26 — TPM2_ReadPublic Response .....	61
Table 27 — TPM2_ActivateCredential Command .....	64
Table 28 — TPM2_ActivateCredential Response .....	64
Table 29 — TPM2_MakeCredential Command .....	68
Table 30 — TPM2_MakeCredential Response .....	68
Table 31 — TPM2_Unseal Command .....	71
Table 32 — TPM2_Unseal Response .....	71
Table 33 — TPM2_ObjectChangeAuth Command .....	74
Table 34 — TPM2_ObjectChangeAuth Response .....	74
Table 35 — TPM2_Duplicate Command .....	78
Table 36 — TPM2_Duplicate Response .....	78
Table 37 — TPM2_Rewrap Command .....	82
Table 38 — TPM2_Rewrap Response .....	82



Table 39 — TPM2_Import Command .....	87
Table 40 — TPM2_Import Response .....	87
Table 41 — Padding Scheme Selection .....	91
Table 42 — Message Size Limits Based on Padding .....	92
Table 43 — TPM2_RSA_Encrypt Command.....	93
Table 44 — TPM2_RSA_Encrypt Response .....	93
Table 45 — TPM2_RSA_Decrypt Command .....	97
Table 46 — TPM2_RSA_Decrypt Response.....	97
Table 47 — TPM2_ECDH_KeyGen Command.....	101
Table 48 — TPM2_ECDH_KeyGen Response .....	101
Table 49 — TPM2_ECDH_ZGen Command.....	104
Table 50 — TPM2_ECDH_ZGen Response .....	104
Table 51 — TPM2_ECC_Parameters Command.....	106
Table 52 — TPM2_ECC_Parameters Response .....	106
Table 53 — TPM2_ZGen_2Phase Command.....	109
Table 54 — TPM2_ZGen_2Phase Response .....	109
Table 55 — Symmetric Chaining Process .....	113
Table 56 — TPM2_EncryptDecrypt Command.....	115
Table 57 — TPM2_EncryptDecrypt Response.....	115
Table 58 — TPM2_Hash Command.....	119
Table 59 — TPM2_Hash Response .....	119
Table 60 — TPM2_HMAC Command.....	122
Table 61 — TPM2_HMAC Response .....	122
Table 62 — TPM2_GetRandom Command.....	126
Table 63 — TPM2_GetRandom Response .....	126
Table 64 — TPM2_StirRandom Command .....	129
Table 65 — TPM2_StirRandom Response.....	129
Table 66 — Hash Selection Matrix .....	131
Table 67 — TPM2_HMAC_Start Command.....	132
Table 68 — TPM2_HMAC_Start Response .....	132
Table 69 — TPM2_HashSequenceStart Command.....	136
Table 70 — TPM2_HashSequenceStart Response .....	136
Table 71 — TPM2_SequenceUpdate Command .....	139
Table 72 — TPM2_SequenceUpdate Response.....	139
Table 73 — TPM2_SequenceComplete Command .....	143
Table 74 — TPM2_SequenceComplete Response.....	143
Table 75 — TPM2_EventSequenceComplete Command .....	147
Table 76 — TPM2_EventSequenceComplete Response.....	147
Table 77 — TPM2_Certify Command.....	153

Table 78 — TPM2_Certify Response .....	153
Table 79 — TPM2_CertifyCreation Command .....	157
Table 80 — TPM2_CertifyCreation Response.....	157
Table 81 — TPM2_Quote Command .....	161
Table 82 — TPM2_Quote Response .....	161
Table 83 — TPM2_GetSessionAuditDigest Command .....	165
Table 84 — TPM2_GetSessionAuditDigest Response .....	165
Table 85 — TPM2_GetCommandAuditDigest Command .....	169
Table 86 — TPM2_GetCommandAuditDigest Response.....	169
Table 87 — TPM2_GetTime Command .....	173
Table 88 — TPM2_GetTime Response .....	173
Table 89 — TPM2_Commit Command.....	179
Table 90 — TPM2_Commit Response .....	179
Table 91 — TPM2_EC_Ephemeral Command.....	184
Table 92 — TPM2_EC_Ephemeral Response .....	184
Table 93 — TPM2_VerifySignature Command.....	187
Table 94 — TPM2_VerifySignature Response .....	187
Table 95 — TPM2_Sign Command .....	191
Table 96 — TPM2_Sign Response .....	191
Table 97 — TPM2_SetCommandCodeAuditStatus Command.....	196
Table 98 — TPM2_SetCommandCodeAuditStatus Response .....	196
Table 99 — TPM2_PCR_Extend Command .....	201
Table 100 — TPM2_PCR_Extend Response.....	201
Table 101 — TPM2_PCR_Event Command .....	204
Table 102 — TPM2_PCR_Event Response.....	204
Table 103 — TPM2_PCR_Read Command.....	207
Table 104 — TPM2_PCR_Read Response .....	207
Table 105 — TPM2_PCR_Allocate Command.....	210
Table 106 — TPM2_PCR_Allocate Response .....	210
Table 107 — TPM2_PCR_SetAuthPolicy Command.....	213
Table 108 — TPM2_PCR_SetAuthPolicy Response .....	213
Table 109 — TPM2_PCR_SetAuthValue Command .....	216
Table 110 — TPM2_PCR_SetAuthValue Response.....	216
Table 111 — TPM2_PCR_Reset Command .....	219
Table 112 — TPM2_PCR_Reset Response.....	219
Table 113 — TPM2_PolicySigned Command .....	235
Table 114 — TPM2_PolicySigned Response.....	235
Table 115 — TPM2_PolicySecret Command .....	240
Table 116 — TPM2_PolicySecret Response.....	240

Table 117 — TPM2\_PolicyTicket Command ..... 244

Table 118 — TPM2\_PolicyTicket Response ..... 244

Table 119 — TPM2\_PolicyOR Command ..... 248

Table 120 — TPM2\_PolicyOR Response..... 248

Table 121 — TPM2\_PolicyPCR Command ..... 251

Table 122 — TPM2\_PolicyPCR Response ..... 251

Table 123 — TPM2\_PolicyLocality Command ..... 255

Table 124 — TPM2\_PolicyLocality Response..... 255

Table 125 — TPM2\_PolicyNV Command..... 259

Table 126 — TPM2\_PolicyNV Response ..... 259

Table 127 — TPM2\_PolicyCounterTimer Command ..... 264

Table 128 — TPM2\_PolicyCounterTimer Response ..... 264

Table 129 — TPM2\_PolicyCommandCode Command ..... 269

Table 130 — TPM2\_PolicyCommandCode Response..... 269

Table 131 — TPM2\_PolicyPhysicalPresence Command..... 272

Table 132 — TPM2\_PolicyPhysicalPresence Response ..... 272

Table 133 — TPM2\_PolicyCpHash Command..... 275

Table 134 — TPM2\_PolicyCpHash Response ..... 275

Table 135 — TPM2\_PolicyNameHash Command..... 279

Table 136 — TPM2\_PolicyNameHash Response ..... 279

Table 137 — TPM2\_PolicyDuplicationSelect Command..... 282

Table 138 — TPM2\_PolicyDuplicationSelect Response ..... 282

Table 139 — TPM2\_PolicyAuthorize Command ..... 286

Table 140 — TPM2\_PolicyAuthorize Response ..... 286

Table 141 — TPM2\_PolicyAuthValue Command ..... 290

Table 142 — TPM2\_PolicyAuthValue Response ..... 290

Table 143 — TPM2\_PolicyPassword Command..... 293

Table 144 — TPM2\_PolicyPassword Response ..... 293

Table 145 — TPM2\_PolicyGetDigest Command..... 296

Table 146 — TPM2\_PolicyGetDigest Response ..... 296

Table 147 — TPM2\_CreatePrimary Command..... 299

Table 148 — TPM2\_CreatePrimary Response ..... 299

Table 149 — TPM2\_HierarchyControl Command ..... 303

Table 150 — TPM2\_HierarchyControl Response ..... 303

Table 151 — TPM2\_SetPrimaryPolicy Command..... 307

Table 152 — TPM2\_SetPrimaryPolicy Response ..... 307

Table 153 — TPM2\_ChangePPS Command ..... 310

Table 154 — TPM2\_ChangePPS Response..... 310

Table 155 — TPM2\_ChangeEPS Command ..... 313

Table 156 — TPM2_ChangeEPS Response.....	313
Table 157 — TPM2_Clear Command.....	316
Table 158 — TPM2_Clear Response .....	316
Table 159 — TPM2_ClearControl Command.....	320
Table 160 — TPM2_ClearControl Response .....	320
Table 161 — TPM2_HierarchyChangeAuth Command.....	323
Table 162 — TPM2_HierarchyChangeAuth Response .....	323
Table 163 — TPM2_DictionaryAttackLockReset Command.....	326
Table 164 — TPM2_DictionaryAttackLockReset Response .....	326
Table 165 — TPM2_DictionaryAttackParameters Command .....	329
Table 166 — TPM2_DictionaryAttackParameters Response.....	329
Table 167 — TPM2_PP_Commands Command.....	332
Table 168 — TPM2_PP_Commands Response .....	332
Table 169 — TPM2_SetAlgorithmSet Command .....	335
Table 170 — TPM2_SetAlgorithmSet Response.....	335
Table 171 — TPM2_FieldUpgradeStart Command.....	340
Table 172 — TPM2_FieldUpgradeStart Response .....	340
Table 173 — TPM2_FieldUpgradeData Command.....	343
Table 174 — TPM2_FieldUpgradeData Response .....	343
Table 175 — TPM2_FirmwareRead Command.....	346
Table 176 — TPM2_FirmwareRead Response .....	346
Table 177 — TPM2_ContextSave Command.....	349
Table 178 — TPM2_ContextSave Response .....	349
Table 179 — TPM2_ContextLoad Command.....	354
Table 180 — TPM2_ContextLoad Response .....	354
Table 181 — TPM2_FlushContext Command.....	359
Table 182 — TPM2_FlushContext Response .....	359
Table 183 — TPM2_EvictControl Command.....	363
Table 184 — TPM2_EvictControl Response .....	363
Table 185 — TPM2_ReadClock Command.....	367
Table 186 — TPM2_ReadClock Response .....	367
Table 187 — TPM2_ClockSet Command.....	370
Table 188 — TPM2_ClockSet Response .....	370
Table 189 — TPM2_ClockRateAdjust Command.....	373
Table 190 — TPM2_ClockRateAdjust Response .....	373
Table 191 — TPM2_GetCapability Command.....	379
Table 192 — TPM2_GetCapability Response .....	379
Table 193 — TPM2_TestParms Command.....	384
Table 194 — TPM2_TestParms Response .....	384

Table 195 — TPM2\_NV\_DefineSpace Command ..... 390

Table 196 — TPM2\_NV\_DefineSpace Response ..... 390

Table 197 — TPM2\_NV\_UndefineSpace Command ..... 395

Table 198 — TPM2\_NV\_UndefineSpace Response ..... 395

Table 199 — TPM2\_NV\_UndefineSpaceSpecial Command..... 398

Table 200 — TPM2\_NV\_UndefineSpaceSpecial Response ..... 398

Table 201 — TPM2\_NV\_ReadPublic Command..... 401

Table 202 — TPM2\_NV\_ReadPublic Response ..... 401

Table 203 — TPM2\_NV\_Write Command..... 404

Table 204 — TPM2\_NV\_Write Response ..... 404

Table 205 — TPM2\_NV\_Increment Command ..... 408

Table 206 — TPM2\_NV\_Increment Response..... 408

Table 207 — TPM2\_NV\_Extend Command..... 412

Table 208 — TPM2\_NV\_Extend Response ..... 412

Table 209 — TPM2\_NV\_SetBits Command..... 416

Table 210 — TPM2\_NV\_SetBits Response ..... 416

Table 211 — TPM2\_NV\_WriteLock Command ..... 420

Table 212 — TPM2\_NV\_WriteLock Response..... 420

Table 213 — TPM2\_NV\_GlobalWriteLock Command..... 424

Table 214 — TPM2\_NV\_GlobalWriteLock Response ..... 424

Table 215 — TPM2\_NV\_Read Command..... 427

Table 216 — TPM2\_NV\_Read Response ..... 427

Table 217 — TPM2\_NV\_ReadLock Command ..... 430

Table 218 — TPM2\_NV\_ReadLock Response ..... 430

Table 219 — TPM2\_NV\_ChangeAuth Command ..... 433

Table 220 — TPM2\_NV\_ChangeAuth Response ..... 433

Table 221 — TPM2\_NV\_Certify Command..... 436

Table 222 — TPM2\_NV\_Certify Response ..... 436



## Trusted Platform Module Library Part 3: Commands

### 1 Scope

This part 3 of the *Trusted Module Library* specification contains the definitions of the TPM commands. These commands make use of the constants, flags, structure, and union definitions defined in part 2: *Structures*.

The detailed description of the operation of the commands is written in the C language with extensive comments. The behavior of the C code in this part 3 is normative but does not fully describe the behavior of a TPM. The combination of this part 3 and part 4: *Supporting Routines* is sufficient to fully describe the required behavior of a TPM.

The code in parts 3 and 4 is written to define the behavior of a compliant TPM. In some cases (e.g., firmware update), it is not possible to provide a compliant implementation. In those cases, any implementation provided by the vendor that meets the general description of the function provided in part 3 would be compliant.

The code in parts 3 and 4 is not written to meet any particular level of conformance nor does this specification require that a TPM meet any particular level of conformance.

### 2 Terms and Definitions

For the purposes of this document, the terms and definitions given in part 1 of this specification apply.

### 3 Symbols and abbreviated terms

For the purposes of this document, the symbols and abbreviated terms given in part 1 apply.

### 4 Notation

#### 4.1 Introduction

In addition to the notation in this clause, the “Notations” clause in Part 1 of this specification is applicable to this Part 3.

Command and response tables used various decorations to indicate the fields of the command and the allowed types. These decorations are described in this clause.

#### 4.2 Table Decorations

The symbols and terms in the Notation column of Table 1 are used in the tables for the command schematics. These values indicate various qualifiers for the parameters or descriptions with which they are associated.

Table 1 — Command Modifiers and Decoration



Notation	Meaning
+	A Type decoration – When appended to a value in the Type column of a command, this symbol indicates that the parameter is allowed to use the “null” value of the data type (see “Conditional Types” in Part 2). The null value is usually TPM_RH_NULL for a handle or TPM_ALG_NULL for an algorithm selector.
@	A Name decoration – When this symbol precedes a handle parameter in the “Name” column, it indicates that an authorization session is required for use of the entity associated with the handle. If a handle does not have this symbol, then an authorization session is not allowed.
+PP	A Description modifier – This modifier may follow TPM_RH_PLATFORM in the “Description” column to indicate that Physical Presence is required when <i>platformAuth/platformPolicy</i> is provided.
+{PP}	A Description modifier – This modifier may follow TPM_RH_PLATFORM to indicate that Physical Presence may be required when <i>platformAuth/platformPolicy</i> is provided. The commands with this notation may be in the <i>setList</i> or <i>clearList</i> of TPM2_PP_Commands().
{NV}	A Description modifier – This modifier may follow the <i>commandCode</i> in the “Description” column to indicate that the command may result in an update of NV memory and be subject to rate throttling by the TPM. If the command code does not have this notation, then the command will, under normal circumstance, not cause a write to NV memory.
{F}	A Description modifier – This modifier indicates that the “flushed” attribute will be SET in the TPMA_CC for the command. The modifier may follow the <i>commandCode</i> in the “Description” column to indicate that any transient handle context used by the command will be flushed from the TPM when the command completes. This may be combined with the {NV} modifier but not with the {E} modifier. EXAMPLE 1 {NV F} EXAMPLE 2 TPM2_SequenceComplete() will flush the context associated with the <i>sequenceHandle</i> .
{E}	A Description modifier – This modifier indicates that the “extensive” attribute will be SET in the TPMA_CC for the command. This modifier may follow the <i>commandCode</i> in the “Description” column to indicate that the command may flush many objects and re-enumeration of the loaded context likely will be required. This may be combined with the {NV} modifier but not with the {F} modifier. EXAMPLE 1 {NV E} EXAMPLE 2 TPM2_Clear() will flush all contexts associated with the Storage hierarchy and the Endorsement hierarchy.
Auth Index:	A Description modifier – When a handle has a “@” decoration, the “Description” column will contain an “Auth Index:” entry for the handle. This entry indicates the number of the authorization session. The authorization sessions associated with handles will occur in the session area in the order of the handles with the “@” modifier. Sessions used only for encryption/decryption or only for audit will follow the handles used for authorization.
Auth Role:	A Description modifier – This will be in the “Description” column of a handle with the “@” decoration. It may have a value of USER, ADMIN or DUP. If the handle has the Auth Role of USER and the handle is an Object, the type of authorization is determined by the setting of <i>userWithAuth</i> in the Object’s attributes. If the Auth Role is ADMIN and the handle is an Object, the type of authorization is determined by the setting of <i>adminWithPolicy</i> in the Object’s attributes. If the DUP role is selected, authorization may only be with a policy session (DUP role only applies to Objects). When either ADMIN or DUP role is selected, a policy command that selects the command being authorized is required to be part of the policy. EXAMPLE TPM2_Certify requires the ADMIN role for the first handle ( <i>objectHandle</i> ). The policy authorization for <i>objectHandle</i> is required to contain TPM2_PolicyCommandCode( <i>commandCode</i> == TPM_CC_Certify). This sets the state of the policy so that it can be used for ADMIN role authorization in TPM2_Certify().  If the handle references an NV Index, then the allowed authorizations are determined by the settings of the attributes of the NV Index as described in Part 2, “TPMA_NV (NV Index Attributes).”



### 4.3 Handle and Parameter Demarcation

The demarcations between the header, handle, and parameter parts are indicated by:

**Table 2 — Separators**

Separator	Meaning
	the values immediately following are in the handle area
	the values immediately following are in the parameter area

### 4.4 AuthorizationSize and ParameterSize

Authorization sessions are not shown in the command or response schematics. When the tag of a command or response is TPM\_ST\_SESSIONS, then a 32-bit value will be present in the command/response buffer to indicate the size of the authorization field or the parameter field. This value shall immediately follow the handle area (which may contain no handles). For a command, this value (*authorizationSize*) indicates the size of the Authorization Area and shall have a value of 9 or more. For a response, this value (*parameterSize*) indicates the size of the parameter area and may have a value of zero.

If the *authorizationSize* field is present in the command, *parameterSize* will be present in the response, but only if the *responseCode* is TPM\_RC\_SUCCESS.

When the command tag is TPM\_ST\_NO\_SESSIONS, no authorizations are present and no *authorizationSize* field is required and shall not be present.

## 5 Normative References

The “Normative References” clause in Part 1 of this specification is applicable to this Part 3.

## 6 Symbols and Abbreviated Terms

The “Symbols and Abbreviated Terms” clause in Part 1 of this specification is applicable to this Part 3.

## 7 Command Processing

### 7.1 Introduction

This clause defines the command validations that are required of any implementation and the response code returned if the indicated check fails. Unless stated otherwise, the order of the checks is not normative and different TPM may give different responses when a command has multiple errors.

In the description below, some statements that describe a check may be followed by a response code in parentheses. This is the normative response code should the indicated check fail. A normative response code may also be included in the statement.

### 7.2 Command Header Validation

Before a TPM may begin the actions associated with a command, a set of command format and consistency checks shall be performed. These checks are listed below and should be performed in the indicated order.

- a) The TPM shall successfully unmarshal a `TPMI_ST_COMMAND_TAG` and verify that it is either `TPM_ST_SESSIONS` or `TPM_ST_NO_SESSIONS` (`TPM_RC_BAD_TAG`).
- b) The TPM shall successfully unmarshal a `UINT32` as the *commandSize*. If the TPM has an interface buffer that is loaded by some hardware process, the number of octets in the input buffer reported by the hardware process shall exactly match the value in *commandSize* (`TPM_RC_COMMAND_SIZE`).

NOTE                    A TPM may have direct access to system memory and unmarshal directly from that memory.

- c) The TPM shall successfully unmarshal a `TPM_CC` and verify that the command is implemented (`TPM_RC_COMMAND_CODE`).

### 7.3 Mode Checks

The following mode checks shall be performed in the order listed:

- a) If the TPM is in Failure mode, then the *commandCode* is TPM\_CC\_GetTestResult or TPM\_CC\_GetCapability (TPM\_RC\_FAILURE) and the command *tag* is TPM\_ST\_NO\_SESSIONS (TPM\_RC\_FAILURE).

NOTE 1 In Failure mode, the TPM has no cryptographic capability and processing of sessions is not supported.

- b) The TPM is in Field Upgrade mode (FUM), the *commandCode* is TPM\_CC\_FieldUpgradeData (TPM\_RC\_UPGRADE).
- c) If the TPM has not been initialized (TPM2\_Startup()), then the *commandCode* is TPM\_CC\_Startup (TPM\_RC\_INITIALIZE).

NOTE 2 The TPM may enter Failure mode during \_TPM\_Init processing. If so, the TPM may process TPM2\_GetTestResult() or TPM2\_GetCapability() before TPM2\_Startup(). Since the platform firmware cannot know that the TPM is in Failure mode without accessing it, and since the first command is required to be TPM2\_Startup(), the expected sequence will be that platform firmware (the CRTM) will issue TPM2\_Startup() and receive TPM\_RC\_FAILURE indicating that the TPM is in Failure mode.

The mode checks may be performed before or after the command header validation.

#### 7.4 Handle Area Validation

After successfully unmarshaling and validating the command header, the TPM shall perform the following checks on the handles and sessions. These checks may be performed in any order.

- a) The TPM shall successfully unmarshal the number of handles required by the command and validate that the value of the handle is consistent with the command syntax. If not, the TPM shall return TPM\_RC\_VALUE.

NOTE 1 The TPM may unmarshal a handle and validate that it references an entity on the TPM before unmarshaling a subsequent handle.

NOTE 2 If the submitted command contains fewer handles than required by the syntax of the command, the TPM may continue to read into the next area and attempt to interpret the data as a handle.

- b) For all handles in the handle area of the command, the TPM will validate that the referenced entity is present in the TPM.
- 1) If the handle references a transient object, the handle shall reference a loaded object (TPM\_RC\_REFERENCE\_H0 + N where N is the number of the NV Index of the handle in the command).

NOTE 3 If the hierarchy for a transient object is disabled, then the transient objects will be flushed so this check will fail.

- 2) If the handle references a persistent object, then
- i) the handle shall reference a persistent object that is currently in TPM non-volatile memory (TPM\_RC\_HANDLE);
  - ii) the hierarchy associated with the object is not disabled (TPM\_RC\_HIERARCHY); and
  - iii) if the TPM implementation moves a persistent object to RAM for command processing then sufficient RAM space is available (TPM\_RC\_OBJECT\_MEMORY).
- 3) If the handle references an NV Index, then
- i) an Index exists that corresponds to the handle (TPM\_RC\_HANDLE); and
  - ii) the hierarchy associated with the NV Index is not disabled (TPM\_RC\_HIERARCHY).

- 4) If the handle references a session, then the session context shall be present in TPM memory (TPM\_RC\_HANDLE).
- 5) If the handle references a primary seed for a hierarchy (TPM\_RH\_ENDORSEMENT, TPM\_RH\_OWNER, or TPM\_RH\_PLATFORM) then the enable for the hierarchy is SET (TPM\_RC\_HIERARCHY).
- 6) If the handle references a PCR, then the value is within the range of PCR supported by the TPM (TPM\_RC\_VALUE)

NOTE 4 In the reference implementation, this TPM\_RC\_VALUE is returned by the unmarshaling code for a TPMI\_DH\_PCR.

## 7.5 Session Area Validation

- a) If the tag is TPM\_ST\_SESSIONS and the command is a context management command (TPM2\_ContextSave(), TPM2\_ContextLoad(), or TPM2\_FlushContext()) the TPM will return TPM\_RC\_AUTH\_CONTEXT.
- b) If the tag is TPM\_ST\_SESSIONS, the TPM will attempt to unmarshal an *authorizationSize* and return TPM\_RC\_AUTHSIZE if the value is not within an acceptable range.
  - 1) The minimum value is (sizeof(TPM\_HANDLE) + sizeof(UINT16) + sizeof(TPMA\_SESSION) + sizeof(UINT16)).
  - 2) The maximum value of *authorizationSize* is equal to  $\text{commandSize} - (\text{sizeof(TPM\_ST)} + \text{sizeof(UINT32)} + \text{sizeof(TPM\_CC)} + (N * \text{sizeof(TPM\_HANDLE)}) + \text{sizeof(UINT32)})$  where N is the number of handles associated with the *commandCode* and may be zero.

NOTE 1 (sizeof(TPM\_ST) + sizeof(UINT32) + sizeof(TPM\_CC)) is the size of a command header. The last UINT32 contains the *authorizationSize* octets, which are not counted as being in the authorization session area.

- c) The TPM will unmarshal the authorization sessions and perform the following validations:
  - 1) If the session handle is not a handle for an HMAC session, a handle for a policy session, or, TPM\_RS\_PW then the TPM shall return TPM\_RC\_HANDLE.
  - 2) If the session is not loaded, the TPM will return the warning TPM\_RC\_REFERENCE\_S0 + N where N is the number of the session (starting at 1).

NOTE 2 If the HMAC and policy session contexts use the same memory, the type of the context must match the type of the handle.

- 3) If the maximum allowed number of sessions have been unmarshaled and fewer octets than indicated in *authorizationSize* were unmarshaled (that is, *authorizationSize* is too large), the TPM shall return TPM\_RC\_AUTHSIZE.
- 4) The consistency of the authorization session attributes is checked.
  - i) An authorization session is present for each of the handles with the “@” decoration (TPM\_RC\_AUTH\_MISSING).
  - ii) Only one session is allowed for:
    - (a) session auditing (TPM\_RC\_ATTRIBUTES) – this session may be used for encrypt or decrypt but may not be a session that is also used for authorization;
    - (b) decrypting a command parameter (TPM\_RC\_ATTRIBUTES) – this may any of the authorization sessions, or the audit session or a session may be added for the single purpose of decrypting a command parameter as long as the total number of sessions does not exceed three; and

- (c) encrypting a response parameter (TPM\_RC\_ATTRIBUTES) – this may be any of the authorization sessions or the audit session if present and a session may be added for the single purpose of encrypting a response parameter as long as the total number of sessions does not exceed three.

NOTE 3            A session used for decrypting a command parameter may also be used for encrypting a response parameter.

## 7.6 Authorization Checks

After unmarshaling and validating the handles and the consistency of the authorization sessions, the authorizations shall be checked. Authorization checks only apply to handles if the handle in the command schematic has the “@” decoration.

- a) The public and sensitive portions of the object shall be present on the TPM (TPM\_RC\_AUTH\_UNAVAILABLE).
- b) If the associated handle is TPM\_RH\_PLATFORM, and the command requires confirmation with physical presence, then physical presence is asserted (TPM\_RC\_PP).
- c) If the object or NV Index is subject to DA protection, and the authorization is with an HMAC or password, then the TPM is not in lockout (TPM\_RC\_LOCKOUT).

NOTE 1            An object is subject to DA protection if its *noDA* attribute is CLEAR. An NV Index is subject to DA protection if its TPMA\_NV\_NO\_DA attribute is CLEAR.

NOTE 2            An HMAC or password is required in a policy session when the policy contains TPM2\_PolicyAuthValue() or TPM2\_PolicyPassword().

- d) If the command requires a handle to have DUP role authorization, then the associated authorization session is a policy session (TPM\_RC\_POLICY\_FAIL).
- e) If the command requires a handle to have ADMIN role authorization:
  - 1) If the entity being authorized is an object and its *adminWithPolicy* attribute is SET, then the authorization session is a policy session (TPM\_RC\_POLICY\_FAIL).

NOTE 3            If *adminWithPolicy* is CLEAR, then any type of authorization session is allowed.

- 2) If the entity being authorized is an NV Index, then the associated authorization session is a policy session.

NOTE 4            The only commands that are currently defined that required use of ADMIN role authorization are commands that operate on objects and NV Indices.

- f) If the command requires a handle to have USER role authorization:
  - 1) If the entity being authorized is an object and its *userWithAuth* attribute is CLEAR, then the associated authorization session is a policy session (TPM\_RC\_POLICY\_FAIL).
  - 2) If the entity being authorized is an NV Index;
    - i) if the authorization session is a policy session;
      - (a) the TPMA\_NV\_POLICYWRITE attribute of the NV Index is SET if the command modifies the NV Index data (TPM\_RC\_AUTH\_UNAVAILABLE);
      - (b) the TPMA\_NV\_POLICYREAD attribute of the NV Index is SET if the command reads the NV Index data (TPM\_RC\_AUTH\_UNAVAILABLE);
    - ii) if the authorization is an HMAC session or a password;

- (a) the TPMA\_NV\_AUTHWRITE attribute of the NV Index is SET if the command modifies the NV Index data (TPM\_RC\_AUTH\_UNAVAILABLE);
  - (b) the TPMA\_NV\_AUTHREAD attribute of the NV Index is SET if the command reads the NV Index data (TPM\_RC\_AUTH\_UNAVAILABLE).
- g) If the authorization is provided by a policy session, then:
- 1) if *policySession*→*timeOut* has been set, the session shall not have expired (TPM\_RC\_EXPIRED);
  - 2) if *policySession*→*cpHash* has been set, it shall match the *cpHash* of the command (TPM\_RC\_POLICY\_FAIL);
  - 3) if *policySession*→*commandCode* has been set, then *commandCode* of the command shall match (TPM\_RC\_POLICY\_CC);
  - 4) *policySession*→*policyDigest* shall match the *authPolicy* associated with the handle (TPM\_RC\_POLICY\_FAIL);
  - 5) if *policySession*→*pcrUpdateCounter* has been set, then it shall match the value of *pcrUpdateCounter* (TPM\_RC\_PCR\_CHANGED); and
  - 6) if the authorization uses an HMAC, then the HMAC is properly constructed using the *authValue* associated with the handle and/or the session secret (TPM\_RC\_AUTH\_FAIL or TPM\_RC\_BAD\_AUTH).

NOTE 5 For a bound session, if the handle references the object used to initiate the session, then the *authValue* will not be required but proof of knowledge of the session secret is necessary.

NOTE 6 A policy session may require proof of knowledge of the *authValue* of the object being authorized.

If the TPM returns an error other than TPM\_RC\_AUTH\_FAIL then the TPM shall not alter any TPM state. If the TPM return TPM\_RC\_AUTH\_FAIL, then the TPM shall not alter any TPM start other than *lockoutCount*.

NOTE 7 The TPM may decrease *failedTries* regardless of any other processing performed by the TPM. That is, the TPM may exit Lockout mode, regardless of the return code.

## 7.7 Parameter Decryption

If an authorization session has the TPMA\_SESSION.*decrypt* attribute SET, and the command does not allow a command parameter to be encrypted, then the TPM will return TPM\_RC\_ATTRIBUTES. Otherwise, the TPM will decrypt the parameter using the values associated with the session before parsing parameters.

## 7.8 Parameter Unmarshaling

### 7.8.1 Introduction

The detailed actions for each command assume that the input parameters of the command have been unmarshaled into a command-specific structure with the structure defined by the command schematic. Additionally, a response-specific output structure is assumed which will receive the values produced by the detailed actions.

NOTE An implementation is not required to process parameters in this manner or to separate the parameter parsing from the command actions. This method was chosen for the specification so that the normative behavior described by the detailed actions would be clear and unencumbered.

Unmarshaling is the process of processing the parameters in the input buffer and preparing the parameters for use by the command-specific action code. No data movement need take place but it is required that the TPM validate that the parameters meet the requirements of the expected data type as defined in Part 2 of this specification.

### 7.8.2 Unmarshaling Errors

When an error is encountered while unmarshaling a command parameter, an error response code is returned and no command processing occurs. A table defining a data type may have response codes embedded in the table to indicate the error returned when the input value does not match the parameters of the table.

NOTE In the reference implementation, a parameter number is added to the response code so that the offending parameter can be isolated.

In many cases, the table contains no specific response code value and the return code will be determined as defined in Table 3.

Table 3 — Unmarshaling Errors

Response Code	Meaning
TPM_RC_ASYMMETRIC	a parameter that should be an asymmetric algorithm selection does not have a value that is supported by the TPM
TPM_RC_BAD_TAG	a parameter that should be a command tag selection has a value that is not supported by the TPM
TPM_RC_COMMAND_CODE	a parameter that should be a command code does not have a value that is supported by the TPM
TPM_RC_HASH	a parameter that should be a hash algorithm selection does not have a value that is supported by the TPM
TPM_RC_INSUFFICIENT	the input buffer did not contain enough octets to allow unmarshaling of the expected data type;
TPM_RC_KDF	a parameter that should be a key derivation scheme (KDF) selection does not have a value that is supported by the TPM
TPM_RC_KEY_SIZE	a parameter that is a key size has a value that is not supported by the TPM
TPM_RC_MODE	a parameter that should be a symmetric encryption mode selection does not have a value that is supported by the TPM
TPM_RC_RESERVED	a non-zero value was found in a reserved field of an attribute structure (TPMA_)
TPM_RC_SCHEME	a parameter that should be signing or encryption scheme selection does not have a value that is supported by the TPM
TPM_RC_SIZE	the value of a size parameter is larger or smaller than allowed
TPM_RC_SYMMETRIC	a parameter that should be a symmetric algorithm selection does not have a value that is supported by the TPM
TPM_RC_TAG	a parameter that should be a structure tag has a value that is not supported by the TPM
TPM_RC_TYPE	The type parameter of a TPMT_PUBLIC or TPMT_SENSITIVE has a value that is not supported by the TPM
TPM_RC_VALUE	a parameter does not have one of its allowed values

In some commands, a parameter may not be used because of various options of that command. However, the unmarshaling code is required to validate that all parameters have values that are allowed by the Part 2 definition of the parameter type even if that parameter is not used in the command actions.

## 7.9 Command Post Processing

When the code that implements the detailed actions of the command completes, it returns a response code. If that code is not TPM\_RC\_SUCCESS, the post processing code will not update any session or audit data and will return a 10-octet response packet.

If the command completes successfully, the tag of the command determines if any authorization sessions will be in the response. If so, the TPM will encrypt the first parameter of the response if indicated by the authorization attributes. The TPM will then generate a new nonce value for each session and, if appropriate, generate an HMAC.



NOTE 1           The authorization attributes were validated during the session area validation to ensure that only one session was used for parameter encryption of the response and that the command allowed encryption in the response.

NOTE 2           No session nonce value is used for a password authorization but the session data is present.

Additionally, if the command is being audited by Command Audit, the audit digest is updated with the *cpHash* of the command and *rpHash* of the response.

## 8 Response Values

### 8.1 Tag

When a command completes successfully, the *tag* parameter in the response shall have the same value as the *tag* parameter in the command (TPM\_ST\_SESSIONS or TPM\_RC\_NO\_SESSIONS). When a command fails (the *responseCode* is not TPM\_RC\_SUCCESS), then the *tag* parameter in the response shall be TPM\_ST\_NO\_SESSIONS.

A special case exists when the command *tag* parameter is not an allowed value (TPM\_ST\_SESSIONS or TPM\_ST\_NO\_SESSIONS). For this case, it is assumed that the system software is attempting to send a command formatted for a TPM 1.2 but the TPM is not capable of executing TPM 1.2 commands. So that the TPM 1.2 compatible software will have a recognizable response, the TPM sets *tag* to TPM\_ST\_RSP\_COMMAND, *responseSize* to 00 00 00 0A<sub>16</sub> and *responseCode* to TPM\_RC\_BAD\_TAG. This is the same response as the TPM 1.2 fatal error for TPM\_BADTAG.

### 8.2 Response Codes

The normal response for any command is TPM\_RC\_SUCCESS. Any other value indicates that the command did not complete and the state of the TPM is unchanged. An exception to this general rule is that the logic associated with dictionary attack protection is allowed to be modified when an authorization failure occurs.

Commands have response codes that are specific to that command and those response codes are enumerated in the detailed actions of each command. The codes associated with the unmarshaling of parameters are documented Table 3. Another set of response code value are not command specific and indicate a problem that is not specific to the command. That is, if the indicated problem is remedied, the same command could be resubmitted and may complete normally.

The commands that are not command specific are listed and described in Table 4.

The reference code for the command actions may have code that generates specific response codes associated with a specific check but the listing of responses may not have that response code listed.

Table 4 — Command-Independent Response Codes

Response Code	Meaning
TPM_RC_CANCELLED	This response code may be returned by a TPM that supports command cancel. When the TPM receives an indication that the current command should be cancelled, the TPM may complete the command or return this code. If this code is returned, then the TPM state is not changed and the same command may be retried.
TPM_RC_CONTEXT_GAP	This response code can be returned for commands that manage session contexts. It indicates that the gap between the lowest numbered active session and the highest numbered session is at the limits of the session tracking logic. The remedy is to load the session context with the lowest number so that its tracking number can be updated.
TPM_RC_LOCKOUT	This response indicates that authorizations for objects subject to DA protection are not allowed at this time because the TPM is in DA lockout mode. The remedy is to wait or to execute TPM2_DictionaryAttackLockoutReset().
TPM_RC_MEMORY	A TPM may use a common pool of memory for objects, sessions, and other purposes. When the TPM does not have enough memory available to perform the actions of the command, it may return TPM_RC_MEMORY. This indicates that the TPM resource manager may flush either sessions or objects in order to make memory available for the command execution. A TPM may choose to return TPM_RC_OBJECT_MEMORY or TPM_RC_SESSION_MEMORY if it needs contexts of a particular type to be flushed.
TPM_RC_NV_RATE	This response code indicates that the TPM is rate-limiting writes to the NV memory in order to prevent wearout. This response is possible for any command that explicitly writes to NV or commands that incidentally use NV such as a command that uses authorization session that may need to update the dictionary attack logic.
TPM_RC_NV_UNAVAILABLE	This response code is similar to TPM_RC_NV_RATE but indicates that access to NV memory is currently not available and the command is not allowed to proceed until it is. This would occur in a system where the NV memory used by the TPM is not exclusive to the TPM and is a shared system resource.
TPM_RC_OBJECT_HANDLES	This response code indicates that the TPM has exhausted its handle space and no new objects can be loaded unless the TPM is rebooted. This does not occur in the reference implementation because of the way that object handles are allocated. However, other implementations are allowed to assign each object a unique handle each time the object is loaded. A TPM using this implementation would be able to load $2^{24}$ objects before the object space is exhausted.
TPM_RC_OBJECT_MEMORY	This response code can be returned by any command that causes the TPM to need an object 'slot'. The most common case where this might be returned is when an object is loaded (TPM2_Load, TPM2_CreatePrimary(), or TPM2_ContextLoad()). However, the TPM implementation is allowed to use object slots for other reasons. In the reference implementation, the TPM copies a referenced persistent object into RAM for the duration of the command. If all the slots are previously occupied, the TPM may return this value. A TPM is allowed to use object slots for other purposes and return this value. The remedy when this response is returned is for the TPM resource manager to flush a transient object.
TPM_RC_REFERENCE_Hx	This response code indicates that a handle in the handle area of the command is not associated with a loaded object. The value of 'x' is in the range 0 to 6 with a value of 0 indicating the 1 <sup>st</sup> handle and 6 representing the 7 <sup>th</sup> . The TPM resource manager needs to find the correct object and load it. It may then adjust the handle and retry the command.  NOTE Usually, this error indicates that the TPM resource manager has a corrupted database.

Response Code	Meaning
TPM_RC_REFERENCE_Sx	This response code indicates that a handle in the session area of the command is not associated with a loaded session. The value of 'x' is in the range 0 to 6 with a value of 0 indicating the 1 <sup>st</sup> session handle and 6 representing the 7 <sup>th</sup> . The TPM resource manager needs to find the correct session and load it. It may then retry the command. NOTE Usually, this error indicates that the TPM resource manager has a corrupted database.
TPM_RC_RETRY	the TPM was not able to start the command
TPM_RC_SESSION_HANDLES	This response code indicates that the TPM does not have a handle to assign to a new session. This response is only returned by TPM2_StartAuthSession(). It is listed here because the command is not in error and the TPM resource manager can remedy the situation by flushing a session (TPM2_FlushContext()).
TPM_RC_SESSION_MEMORY	This response code can be returned by any command that causes the TPM to need a session 'slot'. The most common case where this might be returned is when a session is loaded (TPM2_StartAuthSession() or TPM2_ContextLoad()). However, the TPM implementation is allowed to use object slots for other purposes. The remedy when this response is returned is for the TPM resource manager to flush a transient object.
TPM_RC_SUCCESS	Normal completion for any command. If the responseCode is TPM_RC_SESSIONS, then the rest of the response has the format indicated in the response schematic. Otherwise, the response is a 10 octet value indicating an error.
TPM_RC_TESTING	This response code indicates that the TPM is performing tests and cannot respond to the request at this time. The command may be retried.
TPM_RC_YIELDED	the TPM has suspended operation on the command; forward progress was made and the command may be retried. See Part 1, "Multi-tasking." NOTE This cannot occur on the reference implementation.

## 9 Implementation Dependent

The actions code for each command makes assumptions about the behavior of various sub-system. There are many possible implementations of the subsystems that would achieve an equivalent results. The actions code is not written to anticipate all possible implementations of the sub-systems. Therefore, it is the responsibility of the implementer to ensure that the necessary changes are made to the actions code when the sub-system behavior changes.

## 10 Detailed Actions Assumptions

### 10.1 Introduction

The C code in the Detailed Actions for each command is written with a set of assumptions about the processing performed before the action code is called and the processing that will be done after the action code completes.

### 10.2 Pre-processing

Before calling the command actions code, the following actions have occurred.

- Verification that the handles in the handle area reference entities that are resident on the TPM.

NOTE                    If a handle is in the parameter portion of the command, the associated entity does not have to be loaded, but the handle is required to be the correct type.

- If use of a handle requires authorization, the Password, HMAC, or Policy session associated with the handle has been verified.
- If a command parameter was encrypted using parameter encryption, it was decrypted before being unmarshaled.
- If the command uses handles or parameters, the calling stack contains a pointer to a data structure (**in**) that holds the unmarshaled values for the handles and commands. If the response has handles or parameters, the calling stack contains a pointer to a data structure (**out**) to hold the handles and parameters generated by the command.
- All parameters of the **in** structure have been validated and meet the requirements of the parameter type as defined in Part 2.
- Space set aside for the **out** structure is sufficient to hold the largest **out** structure that could be produced by the command

### 10.3 Post Processing

When the function implementing the command actions completes,

- response parameters that require parameter encryption will be encrypted after the command actions complete;
- audit and session contexts will be updated if the command response is TPM\_RC\_SUCCESS; and
- the command header and command response parameters will be marshaled to the response buffer.

## 11 Start-up

### 11.1 Introduction

This clause contains the commands used to manage the startup and restart state of a TPM.

### 11.2 `_TPM_Init`

#### 11.2.1 General Description

`_TPM_Init` initializes a TPM.

Initialization actions include testing code required to execute the next expected command. If the TPM is in FUM, the next expected command is `TPM2_FieldUpgradeData()`; otherwise, the next expected command is `TPM2_Startup()`.

NOTE 1 If the TPM performs self-tests after receiving `_TPM_Init()` and the TPM enters Failure mode before receiving `TPM2_Startup()` or `TPM2_FieldUpgradeData()`, then the TPM may be able to accept `TPM2_GetTestResult()` or `TPM2_GetCapability()`.

The means of signaling `_TPM_Init` shall be defined in the platform-specific specifications that define the physical interface to the TPM. The platform shall send this indication whenever the platform starts its boot process and only when the platform starts its boot process.

There shall be no software method of generating this indication that does not also reset the platform and begin execution of the CRTM.

NOTE 2 In the reference implementation, this signal causes an internal flag (*s\_initialized*) to be CLEAR. While this flag is CLEAR, the TPM will only accept the next expected command described above.

### 11.2.2 Detailed Actions

```
1 #include "InternalRoutines.h"
```

This function is used to process a `_TPM_Init()` indication.

```
2 void _TPM_Init(void)
3 {
4     // Initialize crypto engine
5     CryptInitUnits();
6
7     // Initialize NV environment
8     NvPowerOn();
9
10    // Start clock
11    TimePowerOn();
12
13    // Set initialization state
14    TPMInit();
15
16    // Set g_DRTMHandle as unassigned
17    g_DRTMHandle = TPM_RH_UNASSIGNED;
18
19    // No H-CRTM, yet.
20    g_DrtmPreStartup = FALSE;
21
22
23    return;
24 }
```



## 11.3 TPM2\_Startup

### 11.3.1 General Description

TPM2\_Startup() is always preceded by \_TPM\_Init, which is the physical indication that TPM initialization is necessary because of a system-wide reset. TPM2\_Startup() is only valid after \_TPM\_Init. Additional TPM2\_Startup() commands are not allowed after it has completed successfully. If a TPM requires TPM2\_Startup() and another command is received, or if the TPM receives TPM2\_Startup() when it is not required, the TPM shall return TPM\_RC\_INITIALIZE.

NOTE 1 See 11.2.1 for other command options for a TPM supporting field upgrade mode.

NOTE 2 \_TPM\_Hash\_Start, \_TPM\_Hash\_Data, and \_TPM\_Hash\_End are not commands and a platform-specific specification may allow these indications between \_TPM\_Init and TPM2\_Startup().

If in Failure mode the TPM shall accept TPM2\_GetTestResult() and TPM2\_GetCapability() even if TPM2\_Startup() is not completed successfully or processed at all.

A Shutdown/Startup sequence determines the way in which the TPM will operate in response to TPM2\_Startup(). The three sequences are:

- 1) TPM Reset – This is a Startup(CLEAR) preceded by either Shutdown(CLEAR) or no TPM2\_Shutdown(). On TPM Reset, all variables go back to their default initialization state.

NOTE 3 Only those values that are specified as having a default initialization state are changed by TPM Reset. Persistent values that have no default initialization state are not changed by this command. Values such as seeds have no default initialization state and only change due to specific commands.

- 2) TPM Restart – This is a Startup(CLEAR) preceded by Shutdown(STATE). This preserves much of the previous state of the TPM except that PCR and the controls associated with the Platform hierarchy are all returned to their default initialization state;
- 3) TPM Resume – This is a Startup(STATE) preceded by Shutdown(STATE). This preserves the previous state of the TPM including the static Root of Trust for Measurement (S-RTM) PCR and the platform controls other than the *phEnable*.

If a TPM receives Startup(STATE) and that was not preceded by Shutdown(STATE), the TPM shall return TPM\_RC\_VALUE.

If, during TPM Restart or TPM Resume, the TPM fails to restore the state saved at the last Shutdown(STATE), the TPM shall enter Failure Mode and return TPM\_RC\_FAILURE.

On any TPM2\_Startup(),

- *phEnable* shall be SET;
- all transient contexts (objects, sessions, and sequences) shall be flushed from TPM memory;
- TPMS\_TIME\_INFO.*time* shall be reset to zero; and
- use of *lockoutAuth* shall be enabled if *lockoutRecovery* is zero.

Additional actions are performed based on the Shutdown/Startup sequence.

On TPM Reset

- *platformAuth* and *platformPolicy* shall be set to the Empty Buffer,
- tracking data for saved session contexts shall be set to its initial value,
- the object context sequence number is reset to zero,
- a new context encryption key shall be generated,
- TPMS\_CLOCK\_INFO.*restartCount* shall be reset to zero,
- TPMS\_CLOCK\_INFO.*resetCount* shall be incremented,
- the PCR Update Counter shall be clear to zero,
- *shEnable* and *ehEnable* shall be SET, and
- PCR in all banks are reset to their default initial conditions as determined by the relevant platform-specific specification.

NOTE 4 PCR may be initialized any time between `_TPM_Init` and the end of `TPM2_Startup()`. PCR that are preserved by TPM Resume will need to be restored during `TPM2_Startup()`.

NOTE 5 See "InitializingPCR" in Part 1 of this specification for a description of the default initial conditions for a PCR.

#### On TPM Restart

- TPMS\_CLOCK\_INFO.*restartCount* shall be incremented,
- *shEnable* and *ehEnable* shall be SET,
- *platformAuth* and *platformPolicy* shall be set to the Empty Buffer, and
- PCR in all banks are reset to their default initial conditions.
- If a CRTM Event sequence is active, extend the PCR designated by the platform-specific specification.

#### On TPM Resume

- the H-CRTM startup method is the same for this `TPM2_Startup()` as for the previous `TPM2_Startup()`; (`TPM_RC_LOCALITY`)
- TPMS\_CLOCK\_INFO.*restartCount* shall be incremented; and
- PCR that are specified in a platform-specific specification to be preserved on TPM Resume are restored to their saved state and other PCR are set to their initial value as determined by a platform-specific specification.

Other TPM state may change as required to meet the needs of the implementation.

If the *startupType* is `TPM_SU_STATE` and the TPM requires `TPM_SU_CLEAR`, then the TPM shall return `TPM_RC_VALUE`.

NOTE 6 The TPM will require `TPM_SU_CLEAR` when no shutdown was performed or after `Shutdown(STATE)`.

NOTE 7 If *startupType* is neither `TPM_SU_STATE` nor `TPM_SU_CLEAR`, then the unmarshaling code returns `TPM_RC_VALUE`.

## 11.3.2 Command and Response

Table 5 — TPM2\_Startup Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Startup {NV}
TPM_SU	startupType	TPM_SU_CLEAR or TPM_SU_STATE

Table 6 — TPM2\_Startup Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 11.3.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Startup_fp.h"

```

Error Returns	Meaning
TPM_RC_VALUE	start up type is not compatible with previous shutdown sequence

```

3  TPM_RC
4  TPM2_Startup(
5      Startup_In      *in          // IN: input parameter list
6  )
7  {
8      STARTUP_TYPE      startup;
9      TPM_RC            result;
10     BOOL              prevDrtmPreStartup;
11
12     // The command needs NV update. Check if NV is available.
13     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
14     // this point
15     result = NvIsAvailable();
16     if(result != TPM_RC_SUCCESS)
17         return result;
18
19     // Input Validation
20
21     // Read orderly shutdown states from previous power cycle
22     NvReadReserved(NV_ORDERLY, &g_prevOrderlyState);
23
24     // HACK to extract the DRTM startup type associated with the previous shutdown
25     prevDrtmPreStartup = (g_prevOrderlyState == (TPM_SU_STATE + 0x8000));
26     if(prevDrtmPreStartup)
27         g_prevOrderlyState = TPM_SU_STATE;
28
29
30     // if the previous power cycle was shut down with no StateSave command, or
31     // with StateSave command for CLEAR, this cycle can not startup up with
32     // STATE
33     if( ( g_prevOrderlyState == SHUTDOWN_NONE
34         || g_prevOrderlyState == TPM_SU_CLEAR
35         )
36         && in->startupType == TPM_SU_STATE
37     )
38         return TPM_RC_VALUE + RC_Startup_startupType;
39
40     // Internal Date Update
41
42     // Translate the TPM2_ShutDown and TPM2_Startup sequence into the startup
43     // types.
44     if(in->startupType == TPM_SU_CLEAR && g_prevOrderlyState == TPM_SU_STATE)
45     {
46         startup = SU_RESTART;
47         // Read state reset data
48         NvReadReserved(NV_STATE_RESET, &gr);
49     }
50     else if(in->startupType == TPM_SU_STATE && g_prevOrderlyState == TPM_SU_STATE)
51     {
52         // For a resume, the H-CRITM startup method must be the same
53         if(g_DrtmPreStartup != prevDrtmPreStartup)
54             return TPM_RC_LOCALITY;

```

```
55
56     // Read state clear and state reset data
57     NvReadReserved(NV_STATE_CLEAR, &gc);
58     NvReadReserved(NV_STATE_RESET, &gr);
59     startup = SU_RESUME;
60 }
61 else
62 {
63     startup = SU_RESET;
64 }
65
66 // Read persistent data from NV
67 NvReadPersistent();
68
69 // Start up subsystems
70 // Start counters and timers
71 TimeStartup(startup);
72
73 // Start dictionary attack subsystem
74 DASTartup(startup);
75
76 // Enable hierarchies
77 HierarchyStartup(startup);
78
79 // Crypto Startup
80 CryptUtilStartup(startup);
81
82 // Restore/Initialize PCR
83 PCRStartup(startup);
84
85 // Restore/Initialize command audit information
86 CommandAuditStartup(startup);
87
88 // Object context variables
89 if(startup == SU_RESET)
90 {
91     // Reset object context ID to 0
92     gr.objectContextID = 0;
93     // Reset clearCount to 0
94     gr.clearCount= 0;
95 }
96
97 // Initialize object table
98 ObjectStartup();
99
100 // Initialize session table
101 SessionStartup(startup);
102
103 // Initialize index/evict data. This function clear read/write locks
104 // in NV index
105 NvEntityStartup(startup);
106
107 // Initialize the orderly shut down flag for this cycle to SHUTDOWN_NONE.
108 gp.orderlyState = SHUTDOWN_NONE;
109 NvWriteReserved(NV_ORDERLY, &gp.orderlyState);
110
111 // Update TPM internal states if command succeeded.
112 // Record a TPM2_Startup command has been received.
113 TPMRegisterStartup();
114
115 return TPM_RC_SUCCESS;
116
117 }
```

## 11.4 TPM2\_Shutdown

### 11.4.1 General Description

This command is used to prepare the TPM for a power cycle. The *shutdownType* parameter indicates how the subsequent TPM2\_Startup() will be processed.

For a *shutdownType* of any type, the volatile portion of Clock is saved to NV memory and the orderly shutdown indication is SET. NV with the TPMA\_NV\_ORDERLY attribute will be updated.

For a *shutdownType* of TPM\_SU\_STATE, the following additional items are saved:

- tracking information for saved session contexts;
- the session context counter;
- PCR that are designated as being preserved by TPM2\_Shutdown(TPM\_SU\_STATE);
- the PCR Update Counter;
- flags associated with supporting the TPMA\_NV\_WRITESTCLEAR and TPMA\_NV\_READSTCLEAR attributes; and
- the command audit digest and count.

The following items shall not be saved and will not be in TPM memory after the next TPM2\_Startup:

- TPM-memory-resident session contexts;
- TPM-memory-resident transient objects; or
- TPM-memory-resident hash contexts created by TPM2\_HashSequenceStart().

Some values may be either derived from other values or saved to NV memory.

This command saves TPM state but does not change the state other than the internal indication that the context has been saved. The TPM shall continue to accept commands. If a subsequent command changes TPM state saved by this command, then the effect of this command is nullified. That is, after state is modified and if no TPM2\_Shutdown() occurs before the next TPM2\_Startup(), then the next TPM2\_Startup() shall be TPM2\_Startup(CLEAR).

## 11.4.2 Command and Response

Table 7 — TPM2\_Shutdown Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Shutdown {NV}
TPM_SU	shutdownType	TPM_SU_CLEAR or TPM_SU_STATE

Table 8 — TPM2\_Shutdown Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 11.4.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Shutdown_fp.h"

```

Error Returns	Meaning
TPM_RC_TYPE	if PCR bank has been re-configured, a CLEAR StateSave () is required

```

3  TPM_RC
4  TPM2_Shutdown(
5      Shutdown_In      *in           // IN: input parameter list
6  )
7  {
8      TPM_RC          result;
9
10     // The command needs NV update. Check if NV is available.
11     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
12     // this point
13     result = NvIsAvailable();
14     if(result != TPM_RC_SUCCESS) return result;
15
16     // Input Validation
17
18     // If PCR bank has been reconfigured, a CLEAR state save is required
19     if(g_pcrReConfig && in->shutdownType == TPM_SU_STATE)
20         return TPM_RC_TYPE + RC_Shutdown_shutdownType;
21
22     // Internal Data Update
23
24     // PCR private data state save
25     PCRStateSave(in->shutdownType);
26
27     // Save clock
28     NvWriteReserved(NV_CLOCK, &go.clock);
29
30     // Save RAM backed NV index data
31     NvStateSave();
32
33     if(in->shutdownType == TPM_SU_STATE)
34     {
35         // Save STATE_RESET and STATE_CLEAR data
36         NvWriteReserved(NV_STATE_CLEAR, &gc);
37         NvWriteReserved(NV_STATE_RESET, &gr);
38     }
39     else if(in->shutdownType == TPM_SU_CLEAR)
40     {
41         // Save STATE_RESET data
42         NvWriteReserved(NV_STATE_RESET, &gr);
43     }
44
45     // Write orderly shut down state
46     if(in->shutdownType == TPM_SU_CLEAR)
47         gp.orderlyState = TPM_SU_CLEAR;
48     else if(in->shutdownType == TPM_SU_STATE)
49         // This is a complete hack to preserve the state of the H-DRTM across
50         // TPM Resume. If we are doing an orderly shutdown, we will set the MSb of
51         // gp.orderlyState and write it to NV. On the next Startup, we will check
52         // that the state of g_DrtmPreStartup matches the saved value and fail if
53         // not. BTW, after a check of the code, it seems that the only check that
54         // is made of gp.orderlyState is to see if it is SHUTDOWN_NONE. There is no

```



```
55     // check to see if it is TPM_SU_STATE or TPM_SU_CLEAR. This is because what
56     // matters to Startup, is in g_prevOrderlyState.
57     gp.orderlyState = g_DrtmPreStartup ? TPM_SU_STATE + 0x8000 : TPM_SU_STATE;
58     else
59         pAssert(FALSE);
60
61     NvWriteReserved(NV_ORDERLY, &gp.orderlyState);
62
63     return TPM_RC_SUCCESS;
64 }
```

## 12 Testing

### 12.1 Introduction

Compliance to standards for hardware security modules may require that the TPM test its functions before the results that depend on those functions may be returned. The TPM may perform operations using testable functions before those functions have been tested as long as the TPM returns no value that depends on the correctness of the testable function.

**EXAMPLE**      TPM2\_PCR\_Event() may be executed before the hash algorithms have been tested. However, until the hash algorithms have been tested, the contents of a PCR may not be used in any command if that command may result in a value being returned to the TPM user. This means that TPM2\_PCR\_Read() or TPM2\_PolicyPCR() could not complete until the hashes have been checked but other TPM2\_PCR\_Event() commands may be executed even though the operation uses previous PCR values.

If a command is received that requires return of a value that depends on untested functions, the TPM shall test the required functions before completing the command.

Once the TPM has received TPM2\_SelfTest() and before completion of all tests, the TPM is required to return TPM\_RC\_TESTING for any command that uses a function that requires a test.

If a self-test fails at any time, the TPM will enter Failure mode. While in Failure mode, the TPM will return TPM\_RC\_FAILURE for any command other than TPM2\_GetTestResult() and TPM2\_GetCapability(). The TPM will remain in Failure mode until the next \_TPM\_Init.

## 12.2 TPM2\_SelfTest

### 12.2.1 General Description

This command causes the TPM to perform a test of its capabilities. If the *fullTest* is YES, the TPM will test all functions. If *fullTest* = NO, the TPM will only test those functions that have not previously been tested.

If any tests are required, the TPM shall either

- a) return TPM\_RC\_TESTING and begin self-test of the required functions, or

NOTE 1 If *fullTest* is NO, and all functions have been tested, the TPM shall return TPM\_RC\_SUCCESS.

- b) perform the tests and return the test result when complete.

If the TPM uses option a), the TPM shall return TPM\_RC\_TESTING for any command that requires use of a testable function, even if the functions required for completion of the command have already been tested.

NOTE 2 This command may cause the TPM to continue processing after it has returned the response. So that software can be notified of the completion of the testing, the interface should include controls that would allow the TPM to generate an interrupt when the "background" processing is complete. This would be in addition to the interrupt that is expected to be available for signaling normal command completion. It is not necessary that there be two interrupts, but the interface should provide a way to indicate the nature of the interrupt (normal command or deferred command).

## 12.2.2 Command and Response

Table 9 — TPM2\_SelfTest Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_SelfTest {NV}
TPMI_YES_NO	fullTest	YES if full test to be performed NO if only test of untested functions required

Table 10 — TPM2\_SelfTest Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

### 12.2.3 Detailed Actions

```
1 #include "InternalRoutines.h"
2 #include "SelfTest_fp.h"
```

Error Returns	Meaning
TPM_RC_TESTING	self test in process

```
3 TPM_RC
4 TPM2_SelfTest(
5     SelfTest_In    *in           // IN: input parameter list
6 )
7 {
8     // Command Output
9
10    // Call self test function in crypt module
11    return CryptSelfTest(in->fullTest);
12 }
```

## 12.3 TPM2\_IncrementalSelfTest

### 12.3.1 General Description

This command causes the TPM to perform a test of the selected algorithms.

NOTE 1           The *toTest* list indicates the algorithms that software would like the TPM to test in anticipation of future use. This allows tests to be done so that a future commands will not be delayed due to testing.

If *toTest* contains an algorithm that has already been tested, it will not be tested again.

NOTE 2           The only way to force retesting of an algorithm is with `TPM2_SelfTest(fullTest = YES)`.

The TPM will return in *toDoList* a list of algorithms that are yet to be tested. This list is not the list of algorithms that are scheduled to be tested but the algorithms/functions that have not been tested. Only the algorithms on the *toTest* list are scheduled to be tested by this command.

Making *toTest* an empty list allows the determination of the algorithms that remain untested without triggering any testing.

If *toTest* is not an empty list, the TPM shall return `TPM_RC_SUCCESS` for this command and then return `TPM_RC_TESTING` for any subsequent command (including `TPM2_IncrementalSelfTest()`) until the requested testing is complete.

NOTE 3           If *toDoList* is empty, then no additional tests are required and `TPM_RC_TESTING` will not be returned in subsequent commands and no additional delay will occur in a command due to testing.

NOTE 4           If none of the algorithms listed in *toTest* is in the *toDoList*, then no tests will be performed.

If all the parameters in this command are valid, the TPM returns `TPM_RC_SUCCESS` and the *toDoList* (which may be empty).

NOTE 5           An implementation may perform all requested tests before returning `TPM_RC_SUCCESS`, or it may return `TPM_RC_SUCCESS` for this command and then return `TPM_RC_TESTING` for all subsequent commands (including `TPM2_IncrementatSelfTest()`) until the requested tests are complete.

## 12.3.2 Command and Response

Table 11 — TPM2\_IncrementalSelfTest Command

Type	Name	Description
TPML_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_IncrementalSelfTest {NV}
TPML_ALG	toTest	list of algorithms that should be tested

Table 12 — TPM2\_IncrementalSelfTest Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPML_ALG	toDoList	list of algorithms that need testing

### 12.3.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "IncrementalSelfTest_fp.h"
3  TPM_RC
4  TPM2_IncrementalSelfTest(
5      IncrementalSelfTest_In    *in,           // IN: input parameter list
6      IncrementalSelfTest_Out    *out          // OUT: output parameter list
7  )
8  {
9  // Command Output
10
11     // Call incremental self test function in crypt module
12     return CryptIncrementalSelfTest(&in->toTest, &out->toDoList);
13 }
```



## 12.4 TPM2\_GetTestResult

### 12.4.1 General Description

This command returns manufacturer-specific information regarding the results of a self-test and an indication of the test status.

If TPM2\_SelfTest() has not been executed and a testable function has not been tested, *testResult* will be TPM\_RC\_NEEDS\_TEST. If TPM2\_SelfTest() has been received and the tests are not complete, *testResult* will be TPM\_RC\_TESTING. If testing of all functions is complete without functional failures, *testResult* will be TPM\_RC\_SUCCESS. If any test failed, *testResult* will be TPM\_RC\_FAILURE. If the TPM is in Failure mode because of an invalid *startupType* in TPM2\_Startup(), *testResult* will be TPM\_RC\_INITIALIZE.

This command will operate when the TPM is in Failure mode so that software can determine the test status of the TPM and so that diagnostic information can be obtained for use in failure analysis. If the TPM is in Failure mode, then *tag* is required to be TPM\_ST\_NO\_SESSIONS or the TPM shall return TPM\_RC\_FAILURE.

## 12.4.2 Command and Response

Table 13 — TPM2\_GetTestResult Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_GetTestResult

Table 14 — TPM2\_GetTestResult Response

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_MAX_BUFFER	outData	test result data contains manufacturer-specific information
TPM_RC	testResult	

### 12.4.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "GetTestResult_fp.h"
3  TPM_RC
4  TPM2_GetTestResult(
5      GetTestResult_Out    *out           // OUT: output parameter list
6  )
7  {
8  // Command Output
9
10     // Call incremental self test function in crypt module
11     out->testResult = CryptGetTestResult(&out->outData);
12
13     return TPM_RC_SUCCESS;
14 }
```

## 13 Session Commands

### 13.1 TPM2\_StartAuthSession

#### 13.1.1 General Description

This command is used to start an authorization session using alternative methods of establishing the session key (*sessionKey*). The session key is then used to derive values used for authorization and for encrypting parameters.

This command allows injection of a secret into the TPM using either asymmetric or symmetric encryption. The type of *tpmKey* determines how the value in *encryptedSalt* is encrypted. The decrypted secret value is used to compute the *sessionKey*.

NOTE 1 If *tpmKey* is TPM\_RH\_NULL, then *encryptedSalt* is required to be an Empty Buffer.

The label value of “SECRET” (see “Terms and Definitions” in Part 1 of this specification) is used in the recovery of the secret value.

The TPM generates the *sessionKey* from the recovered secret value.

No authorization is required for *tpmKey* or *bind*.

NOTE 2 The justification for using *tpmKey* without providing authorization is that the result of using the key is not available to the caller, except indirectly through the *sessionKey*. This does not represent a point of attack on the value of the key. If the caller attempts to use the session without knowing the *sessionKey* value, it is an authorization failure that will trigger the dictionary attack logic.

The entity referenced with the *handle* parameter contributes an authorization value to the *sessionKey* generation process.

If both *tpmKey* and *handle* are TPM\_ALG\_NULL, then *sessionKey* is set to the Empty Buffer. If *tpmKey* is not TPM\_ALG\_NULL, then *encryptedSecret* is used in the computation of *sessionKey*. If *handle* is not TPM\_ALG\_NULL, the *authValue* of *handle* is used in the *sessionKey* computation.

If *symmetric* specifies a block cipher, then TPM\_ALG\_CFB is the only allowed value for the *mode* field in the parameter (TPM\_RC\_MODE).

This command starts an authorization session and returns the session handle along with an initial *nonceTPM* in the response.

If the TPM does not have a free slot for an authorization session, it shall return TPM\_RC\_SESSION\_HANDLES.

If the TPM implements a “gap” scheme for assigning *contextID* values, then the TPM shall return TPM\_RC\_CONTEXT\_GAP if creating the session would prevent recycling of old saved contexts (See “Context Management” in Part 1).

If *tpmKey* is not TPM\_ALG\_NULL then *salt* shall be a TPM2B\_ENCRYPTED\_SECRET of the proper type for *tpmKey*. The TPM shall return TPM\_RC\_VALUE if:

- a) *tpmKey* references an RSA key and *salt*
  - 1) does not contain a value that is the size of the public modulus of *tpmKey*,
  - 2) has a value that is greater than the public modulus of *tpmKey*,
  - 3) is not a properly encode OAEP value, or
  - 4) the encode value is larger than the size of the digest produced by the *nameAlg* of *tpmKey*, or
- b) *tpmKey* references an ECC key and *encryptedSalt*

- 1) does not contain a `TPMS_ECC_POINT` or
- 2) is not a point on the curve of `tpmKey`;

NOTE 3            When ECC is used, the point multiply process produces a value (Z) that is used in a KDF to produce the final secret value. The size of the secret value is an input parameter to the KDF and the result will be set to be the size of the digest produced by the `nameAlg` of `tpmKey`.

- c) `tpmKey` references a symmetric block cipher or a `keyedHash` object and `encryptedSalt` contains a value that is larger than the size of the digest produced by the `nameAlg` of `tpmKey`.

For all session types, this command will cause initialization of the `sessionKey` and may establish binding between the session and an object (the `bind` object). If `sessionType` is `TPM_SE_POLICY` or `TPM_SE_TRIAL`, the additional session initialization is:

- set `policySession`→`policyDigest` to a Zero Digest (the digest size for `policySession`→`policyDigest` is the size of the digest produced by `authHash`);
- authorization may be given at any locality;
- authorization may apply to any command code;
- authorization may apply to any command parameters or handles;
- the authorization has no time limit;
- an `authValue` is not needed when the authorization is used;
- the session is not bound;
- the session is not an audit session; and
- the time at which the policy session was created is recorded.

Additionally, if `sessionType` is `TPM_SE_TRIAL`, the session will not be usable for authorization but can be used to compute the `authPolicy` for an object.

NOTE 4            Although this command changes the session allocation information in the TPM, it does not invalidate a saved context. That is, `TPM2_Shutdown()` is not required after this command in order to re-establish the orderly state of the TPM. This is because the created context will occupy an available slot in the TPM and sessions in the TPM do not survive any `TPM2_Startup()`. However, if a created session is context saved, the orderly state does change.

The TPM shall return `TPM_RC_SIZE` if `nonceCaller` is less than 16 octets or is greater than the size of the digest produced by `authHash`.

## 13.1.2 Command and Response

Table 15 — TPM2\_StartAuthSession Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_StartAuthSession
TPMI_DH_OBJECT+	tpmKey	handle of a loaded decrypt key used to encrypt <i>salt</i> may be TPM_RH_NULL Auth Index: None
TPMI_DH_ENTITY+	bind	entity providing the <i>authValue</i> may be TPM_RH_NULL Auth Index: None
TPM2B_NONCE	nonceCaller	initial <i>nonceCaller</i> , sets nonce size for the session shall be at least 16 octets
TPM2B_ENCRYPTED_SECRET	encryptedSalt	value encrypted according to the type of <i>tpmKey</i> If <i>tpmKey</i> is TPM_RH_NULL, this shall be the Empty Buffer.
TPM_SE	sessionType	indicates the type of the session; simple HMAC or policy (including a trial policy)
TPMT_SYM_DEF+	symmetric	the algorithm and key size for parameter encryption may select TPM_ALG_NULL
TPMI_ALG_HASH	authHash	hash algorithm to use for the session Shall be a hash algorithm supported by the TPM and not TPM_ALG_NULL

Table 16 — TPM2\_StartAuthSession Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPMI_SH_AUTH_SESSION	sessionHandle	handle for the newly created session
TPM2B_NONCE	nonceTPM	the initial nonce from the TPM, used in the computation of the <i>sessionKey</i>

## 13.1.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "StartAuthSession_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	<i>tpmKey</i> does not reference a decrypt key
TPM_RC_CONTEXT_GAP	the difference between the most recently created active context and the oldest active context is at the limits of the TPM
TPM_RC_HANDLE	input decrypt key handle only has public portion loaded, or input bind point is not a null handle but session to be created is policy session.
TPM_RC_MODE	<i>symmetric</i> specifies a block cipher but the mode is not TPM_ALG_CFB.
TPM_RC_SESSION_HANDLES	no session handle is available
TPM_RC_SESSION_MEMORY	no more slots for loading a session
TPM_RC_SIZE	nonce less than 16 octets or greater than the size of the digest produced by <i>authHash</i>
TPM_RC_VALUE	secret size does not match decrypt key type; or the recovered secret is larger than the digest size of the <i>nameAlg</i> of <i>tpmKey</i> ; or, for an RSA decrypt key, if <i>encryptedSecret</i> is greater than the public exponent of <i>tpmKey</i> .

```

3  TPM_RC
4  TPM2_StartAuthSession(
5      StartAuthSession_In    *in,        // IN: input parameter buffer
6      StartAuthSession_Out  *out        // OUT: output parameter buffer
7  )
8  {
9      TPM_RC                result = TPM_RC_SUCCESS;
10     OBJECT                 *tpmKey;    // TPM key for decrypt salt
11     SESSION                 *session;   // session internal data
12     TPM2B_DATA              salt;
13
14     // Input Validation
15
16     // Check input nonce size.  IT should be at least 16 bytes but not larger
17     // than the digest size of session hash.
18     if( in->nonceCaller.t.size < 16
19         || in->nonceCaller.t.size > CryptGetHashDigestSize(in->authHash))
20         return TPM_RC_SIZE + RC_StartAuthSession_nonceCaller;
21
22     // If an decrypt key is passed in, check its validation
23     if(in->tpmKey != TPM_RH_NULL)
24     {
25         // secret size can not be 0
26         if(in->encryptedSalt.t.size == 0)
27             return TPM_RC_VALUE + RC_StartAuthSession_encryptedSalt;
28
29         // Get pointer to loaded decrypt key
30         tpmKey = ObjectGet(in->tpmKey);
31
32         // Decrypting salt requires accessing the private portion of a key.
33         // Therefore, tpmKey can not be a key with only public portion loaded
34         if(tpmKey->attributes.publicOnly)
35             return TPM_RC_HANDLE + RC_StartAuthSession_tpmKey;

```

```

36
37 // HMAC session input handle check.
38 // tpmKey should be a decryption key
39 if(tpmKey->publicArea.objectAttributes.decrypt != SET)
40     return TPM_RC_ATTRIBUTES + RC_StartAuthSession_tpmKey;
41
42
43 // Secret Decryption. A TPM_RC_VALUE, TPM_RC_KEY or Unmarshal errors
44 // may be returned at this point
45 result = CryptSecretDecrypt(in->tpmKey, &in->nonceCaller, "SECRET",
46                             &in->encryptedSalt, &salt);
47 if(result != TPM_RC_SUCCESS)
48     return TPM_RC_VALUE + RC_StartAuthSession_encryptedSalt;
49
50
51 }
52 else
53 {
54     // secret size must be 0
55     if(in->encryptedSalt.t.size != 0)
56         return TPM_RC_VALUE + RC_StartAuthSession_encryptedSalt;
57     salt.t.size = 0;
58 }
59 // If 'symmetric' is a symmetric block cipher (not TPM_ALG_NULL or TPM_ALG_XOR)
60 // then the mode must be CFB.
61 if( in->symmetric.algorithm != TPM_ALG_NULL
62     && in->symmetric.algorithm != TPM_ALG_XOR
63     && in->symmetric.mode.sym != TPM_ALG_CFB)
64     return TPM_RC_MODE + RC_StartAuthSession_symmetric;
65
66 // Internal Data Update
67
68 // Create internal session structure. TPM_RC_CONTEXT_GAP, TPM_RC_NO_HANDLES
69 // or TPM_RC_SESSION_MEMORY errors may be returned returned at this point.
70 //
71 // The detailed actions for creating the session context are not shown here
72 // as the details are implementation dependent
73 // SessionCreate sets the output handle
74 result = SessionCreate(in->sessionType, in->authHash,
75                       &in->nonceCaller, &in->symmetric,
76                       in->bind, &salt, &out->sessionHandle);
77
78 if(result != TPM_RC_SUCCESS)
79     return result;
80
81 // Command Output
82
83 // Get session pointer
84 session = SessionGet(out->sessionHandle);
85
86 // Copy nonceTPM
87 out->nonceTPM = session->nonceTPM;
88
89 return TPM_RC_SUCCESS;
90 }

```



## 13.2 TPM2\_PolicyRestart

### 13.2.1 General Description

This command allows a policy authorization session to be returned to its initial state. This command is used after the TPM returns TPM\_RC\_PCR\_CHANGED. That response code indicates that a policy will fail because the PCR have changed after TPM2\_PolicyPCR() was executed. Restarting the session allows the authorizations to be replayed, and if the PCR are valid for the policy, the policy may then succeed.

This command does not reset the policy ID or the policy start time.

## 13.2.2 Command and Response

Table 17 — TPM2\_PolicyRestart Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyRestart
TPMI_SH_POLICY	sessionHandle	the handle for the policy session

Table 18 — TPM2\_PolicyRestart Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

### 13.2.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "PolicyRestart_fp.h"
3  TPM_RC
4  TPM2_PolicyRestart(
5      PolicyRestart_In      *in           // IN: input parameter list
6  )
7  {
8      SESSION                *session;
9      BOOL                   wasTrialSession;
10
11     // Internal Data Update
12
13     session = SessionGet(in->sessionHandle);
14     wasTrialSession = session->attributes.isTrialPolicy == SET;
15
16     // Initialize policy session
17     SessionResetPolicyData(session);
18
19     session->attributes.isTrialPolicy = wasTrialSession;
20
21     return TPM_RC_SUCCESS;
22 }
```

## 14 Object Commands

### 14.1 TPM2\_Create

#### 14.1.1 General Description

This command is used to create an object that can be loaded into a TPM using TPM2\_Load(). If the command completes successfully, the TPM will create the new object and return the object's creation data (*creationData*), its public area (*outPublic*), and its encrypted sensitive area (*outPrivate*). Preservation of the returned data is the responsibility of the caller. The object will need to be loaded (TPM2\_Load()) before it may be used.

TPM2B\_PUBLIC template (*inPublic*) contains all of the fields necessary to define the properties of the new object. The setting for these fields is defined in "Public Area Template" in Part 1 and "TPMA\_OBJECT" in Part 2.

The *parentHandle* parameter shall reference a loaded decryption key that has both the public and sensitive area loaded.

When defining the object, the caller provides a template structure for the object in a TPM2B\_PUBLIC structure (*inPublic*), an initial value for the object's *authValue* (*inSensitive.authValue*), and, if the object is a symmetric object, an optional initial data value (*inSensitive.data*). The TPM shall validate the consistency of *inPublic.attributes* according to the Creation rules in "TPMA\_OBJECT" in Part 2.

The methods in this clause are used by both TPM2\_Create() and TPM2\_CreatePrimary(). When a value is indicated as being TPM-generated, the value is filled in by bits from the RNG if the command is TPM2\_Create() and with values from KDFa() if the command is TPM2\_CreatePrimary(). The parameters of each creation value are specified in Part 1.

The *sensitiveDataOrigin* attribute of *inPublic* shall be SET if *inSensitive.data* is an Empty Buffer and CLEAR if *inSensitive.data* is not an Empty Buffer or the TPM shall return TPM\_RC\_ATTRIBUTES.

The TPM will create new data for the sensitive area and compute a TPMT\_PUBLIC.*unique* from the sensitive area based on the object type:

- a) For a symmetric key:
  - 1) If *inSensitive.data* is the Empty Buffer, a TPM-generated key value is placed in the new object's TPMT\_SENSITIVE.*symKey.buffer*. The size of the key will be determined by *inPublic.publicArea.parameters*.
  - 2) If *inSensitive.data* is not the Empty Buffer, the TPM will validate that the size of *inSensitive.data* is no larger than the key size indicated in the *inPublic template* (TPM\_RC\_SIZE) and copy the *inSensitive.data* to TPMT\_SENSITIVE.*symKey.buffer* of the new object.
  - 3) A TPM-generated obfuscation value is placed in TPMT\_SENSITIVE.*sensitive.any.buffer*. The size of the obfuscation value is the size of the digest produced by the *nameAlg* in *inPublic*.
  - 4) The TPMT\_PUBLIC.*unique.sym.buffer* value for the new object is then generated, as shown in equation (1) below, by hashing the key and obfuscation values in the TPMT\_SENSITIVE with the *nameAlg* of the object.

$$unique := H_{nameAlg}(symKey.buffer || sensitive.any.buffer) \quad (1)$$

- b) If the Object is an asymmetric key:
  - 1) If *sensitive.data* is not the Empty Buffer, then the TPM shall return TPM\_RC\_VALUE.
  - 2) A TPM-generated private key value is created with the size determined by the parameters of *inPublic.publicArea.parameters*.

- 3) If the key is a Storage Key, a TPM-generated TPMT\_SENSITIVE.*symKey* value is created; otherwise, TPMT\_SENSITIVE.*symKey.size* is set to zero.
- 4) The public *unique* value is computed from the private key according to the methods of the key type.
- 5) If the key is an ECC key and the scheme required by the *curveID* is not the same as *scheme* in the public area of the template, then the TPM shall return TPM\_RC\_SCHEME.
- 6) If the key is an ECC key and the KDF required by the *curveID* is not the same as *kdf* in the public area of the template, then the TPM shall return TPM\_RC\_KDF.

NOTE 1      There is currently no command in which the caller may specify the KDF to be used with an ECC decryption key. Since there is no use for this capability, the reference implementation requires that the *kdf* in the template be set to TPM\_ALG\_NULL or TPM\_RC\_KDF is returned.

c) If the Object is a keyedHash object:

- 1) If *inSensitive.data* is an Empty Buffer, and neither *sign* nor *decrypt* is SET in *inPublic.attributes*, the TPM shall return TPM\_RC\_ATTRIBUTES.
- 2) If *inSensitive.data* is not an Empty Buffer, the TPM will copy the *inSensitive.data* to TPMT\_SENSITIVE.*sensitive* of the new object.

NOTE 2      The size of *inSensitive.data* is limited to be no larger than the largest value of TPMT\_SENSITIVE.*sensitive.bits.data* by MAX\_SYM\_DATA.

- 3) If *inSensitive.data* is an Empty Buffer, a TPM-generated key value that is the size of the digest produced by the *nameAlg* in *inPublic* is placed in TPMT\_SENSITIVE.*sensitive.any.buffer*.
- 4) A TPM-generated obfuscation value that is half the size of the digest produced by the *nameAlg* of *inPublic* is placed in TPMT\_SENSITIVE.*symKey.buffer*.
- 5) The TPMT\_PUBLIC.*unique.sym.buffer* value for the new object is then generated, as shown in equation (1) above, by hashing the key and obfuscation values in the TPMT\_SENSITIVE with the *nameAlg* of the object.

For TPM2\_Load(), the TPM will apply normal symmetric protections to the created TPMT\_SENSITIVE to create *outPublic*.

NOTE 3      The encryption key is derived from the symmetric seed in the sensitive area of the parent.

In addition to *outPublic* and *outPrivate*, the TPM will build a TPMS\_CREATION\_DATA structure for the object. This structure is returned in *creationData*. Additionally, the digest of this structure is returned in *creationHash*, and, finally, a TPMT\_TK\_CREATION is created so that the association between the creation data and the object may be validated by TPM2\_CertifyCreation().

If the object being created is a Storage Key and *inPublic.objectAttributes.fixedParent* is SET, then the algorithms of *inPublic* are required to match those of the parent. The algorithms that must match are *inPublic.type*, *inPublic.nameAlg*, and *inPublic.parameters*. If *inPublic.type* does not match, the TPM shall return TPM\_RC\_TYPE. If *inPublic.nameAlg* does not match, the TPM shall return TPM\_RC\_HASH. If *inPublic.parameters* does not match, the TPM shall return TPM\_RC\_ASSYMETRIC. The TPM shall not differentiate between mismatches of the components of *inPublic.parameters*.

EXAMPLE      If the *inPublic.parameters.ecc.symmetric.algorithm* does not match the parent, the TPM shall return TPM\_RC\_ASSYMETRIC rather than TPM\_RC\_SYMMETRIC.

The *sensitive* parameter may be encrypted using parameter encryption.

## 14.1.2 Command and Response

Table 19 — TPM2\_Create Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Create
TPMI_DH_OBJECT	@parentHandle	handle of parent for new object Auth Index: 1 Auth Role: USER
TPM2B_SENSITIVE_CREATE	inSensitive	the sensitive data
TPM2B_PUBLIC	inPublic	the public template
TPM2B_DATA	outsideInfo	data that will be included in the creation data for this object to provide permanent, verifiable linkage between this object and some object owner data
TPML_PCR_SELECTION	creationPCR	PCR that will be used in creation data

Table 20 — TPM2\_Create Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PRIVATE	outPrivate	the private portion of the object
TPM2B_PUBLIC	outPublic	the public portion of the created object
TPM2B_CREATION_DATA	creationData	contains a TPMS_CREATION_DATA
TPM2B_DIGEST	creationHash	digest of <i>creationData</i> using <i>nameAlg</i> of <i>outPublic</i>
TPMT_TK_CREATION	creationTicket	ticket used by TPM2_CertifyCreation() to validate that the creation data was produced by the TPM

## 14.1.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Object_spt_fp.h"
3  #include "Create_fp.h"

```

Error Returns	Meaning
TPM_RC_ASYMMETRIC	non-duplicable storage key and its parent have different public params
TPM_RC_ATTRIBUTES	<i>sensitiveDataOrigin</i> is CLEAR when 'sensitive. data' is an Empty Buffer, or is SET when 'sensitive. data' is not empty; <i>fixedTPM</i> , <i>fixedParent</i> , or <i>encryptedDuplication</i> attributes are inconsistent between themselves or with those of the parent object; inconsistent <i>restricted</i> , <i>decrypt</i> and <i>sign</i> attributes; attempt to inject sensitive data for an asymmetric key; attempt to create a symmetric cipher key that is not a decryption key
TPM_RC_HASH	non-duplicable storage key and its parent have different name algorithm
TPM_RC_KDF	incorrect KDF specified for decrypting keyed hash object
TPM_RC_KEY	invalid key size values in an asymmetric key public area
TPM_RC_KEY_SIZE	key size in public area for symmetric key differs from the size in the sensitive creation area; may also be returned if the TPM does not allow the key size to be used for a Storage Key
TPM_RC_SCHEME	inconsistent attributes <i>decrypt</i> , <i>sign</i> , <i>restricted</i> and key's scheme ID; or hash algorithm is inconsistent with the scheme ID for keyed hash object
TPM_RC_SIZE	size of public auth policy or sensitive auth value does not match digest size of the name algorithm sensitive data size for the keyed hash object is larger than is allowed for the scheme
TPM_RC_SYMMETRIC	a storage key with no symmetric algorithm specified; or non-storage key with symmetric algorithm different from TPM_ALG_NULL
TPM_RC_TYPE	unknown object type; non-duplicable storage key and its parent have different types; <i>parentHandle</i> does not reference a restricted decryption key in the storage hierarchy with both public and sensitive portion loaded
TPM_RC_VALUE	exponent is not prime or could not find a prime using the provided parameters for an RSA key; unsupported name algorithm for an ECC key
TPM_RC_OBJECT_MEMORY	there is no free slot for the object. This implementation does not return this error.

```

4  TPM_RC
5  TPM2_Create(
6      Create_In      *in,           // IN: input parameter list
7      Create_Out     *out          // OUT: output parameter list
8  )
9  {
10     TPM_RC          result = TPM_RC_SUCCESS;
11     TPMT_SENSITIVE sensitive;
12     TPM2B_NAME      name;
13
14     // Input Validation
15
16     OBJECT          *parentObject;
17
18     parentObject = ObjectGet(in->parentHandle);

```

```

19
20 // Does parent have the proper attributes?
21 if(!AreAttributesForParent(parentObject))
22     return TPM_RC_TYPE + RC_Create_parentHandle;
23
24 // The sensitiveDataOrigin attribute must be consistent with the setting of
25 // the size of the data object in inSensitive.
26 if( (in->inPublic.t.publicArea.objectAttributes.sensitiveDataOrigin == SET)
27     != (in->inSensitive.t.sensitive.data.t.size == 0))
28     // Mismatch between the object attributes and the parameter.
29     return TPM_RC_ATTRIBUTES + RC_Create_inSensitive;
30
31 // Check attributes in input public area. TPM_RC_ASYMMETRIC, TPM_RC_ATTRIBUTES,
32 // TPM_RC_HASH, TPM_RC_KDF, TPM_RC_SCHEME, TPM_RC_SIZE, TPM_RC_SYMMETRIC,
33 // or TPM_RC_TYPE error may be returned at this point.
34 result = PublicAttributesValidation(FALSE, in->parentHandle,
35                                   &in->inPublic.t.publicArea);
36 if(result != TPM_RC_SUCCESS)
37     return RcSafeAddToResult(result, RC_Create_inPublic);
38
39 // Validate the sensitive area values
40 if( MemoryRemoveTrailingZeros(&in->inSensitive.t.sensitive.userAuth)
41     > CryptGetHashDigestSize(in->inPublic.t.publicArea.nameAlg))
42     return TPM_RC_SIZE + RC_Create_inSensitive;
43
44 // Command Output
45
46 // Create object crypto data
47 result = CryptCreateObject(in->parentHandle, &in->inPublic.t.publicArea,
48                            &in->inSensitive.t.sensitive, &sensitive);
49 if(result != TPM_RC_SUCCESS)
50     return result;
51
52 // Fill in creation data
53 FillInCreationData(in->parentHandle, in->inPublic.t.publicArea.nameAlg,
54                   &in->creationPCR, &in->outsideInfo,
55                   &out->creationData, &out->creationHash);
56
57 // Copy public area from input to output
58 out->outPublic.t.publicArea = in->inPublic.t.publicArea;
59
60 // Compute name from public area
61 ObjectComputeName(&(out->outPublic.t.publicArea), &name);
62
63 // Compute creation ticket
64 TicketComputeCreation(EntityGetHierarchy(in->parentHandle), &name,
65                               &out->creationHash, &out->creationTicket);
66
67 // Prepare output private data from sensitive
68 SensitiveToPrivate(&sensitive, &name, in->parentHandle,
69                  out->outPublic.t.publicArea.nameAlg,
70                  &out->outPrivate);
71
72 return TPM_RC_SUCCESS;
73 }

```



## 14.2 TPM2\_Load

### 14.2.1 General Description

This command is used to load objects into the TPM. This command is used when both a TPM2B\_PUBLIC and TPM2B\_PRIVATE are loaded. If only a TPM2B\_PUBLIC is to be loaded, the TPM2\_LoadExternal command is used.

NOTE 1 Loading an object is not the same as restoring a saved object context.

The object's TPMA\_OBJECT will be checked according to the rules defined in "TPMA\_OBJECT" in Part 2 of this specification.

Objects loaded using this command will have a Name. The Name is the concatenation of *nameAlg* and the digest of the public area using the *nameAlg*.

NOTE 2 *nameAlg* is a parameter in the public area of the *inPublic* structure.

If *inPrivate.size* is zero, the load will fail.

After *inPrivate.buffer* is decrypted using the symmetric key of the parent, the integrity value shall be checked before the sensitive area is used, or unmarshaled.

NOTE 3 Checking the integrity before the data is used prevents attacks on the sensitive area by fuzzing the data and looking at the differences in the response codes.

The command returns a handle for the loaded object and the Name that the TPM computed for *inPublic.public* (that is, the digest of the TPMT\_PUBLIC structure in *inPublic*).

NOTE 4 The TPM-computed Name is provided as a convenience to the caller for those cases where the caller does not implement the hash algorithms specified in the *nameAlg* of the object.

NOTE 5 The returned handle is associated with the object until the object is flushed (TPM2\_FlushContext) or until the next TPM2\_Startup.

For all objects, the size of the key in the sensitive area shall be consistent with the key size indicated in the public area or the TPM shall return TPM\_RC\_KEY\_SIZE.

Before use, a loaded object shall be checked to validate that the public and sensitive portions are properly linked, cryptographically. Use of an object includes use in any policy command. If the parts of the object are not properly linked, the TPM shall return TPM\_RC\_BINDING.

EXAMPLE 1 For a symmetric object, the unique value in the public area shall be the digest of the sensitive key and the obfuscation value.

EXAMPLE 2 For a two-prime RSA key, the remainder when dividing the public modulus by the private key shall be zero and it shall be possible to form a private exponent from the two prime factors of the public modulus.

EXAMPLE 3 For an ECC key, the public point shall be  $f(x)$  where  $x$  is the private key.

## 14.2.2 Command and Response

Table 21 — TPM2\_Load Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Load
TPMI_DH_OBJECT	@parentHandle	TPM handle of parent key; shall not be a reserved handle Auth Index: 1 Auth Role: USER
TPM2B_PRIVATE	inPrivate	the private portion of the object
TPM2B_PUBLIC	inPublic	the public portion of the object

Table 22 — TPM2\_Load Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM_HANDLE	objectHandle	handle for the loaded object
TPM2B_NAME	name	Name of the loaded object

## 14.2.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Load_fp.h"
3  #include "Object_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_ASYMMETRIC	storage key with different asymmetric type than parent
TPM_RC_ATTRIBUTES	<i>inPublic</i> attributes are not allowed with selected parent
TPM_RC_BINDING	<i>inPrivate</i> and <i>inPublic</i> are not cryptographically bound
TPM_RC_HASH	incorrect hash selection for signing key
TPM_RC_INTEGRITY	HMAC on <i>inPrivate</i> was not valid
TPM_RC_KDF	KDF selection not allowed
TPM_RC_KEY	the size of the object's <i>unique</i> field is not consistent with the indicated size in the object's parameters
TPM_RC_OBJECT_MEMORY	no available object slot
TPM_RC_SCHEME	the signing scheme is not valid for the key
TPM_RC_SENSITIVE	the <i>inPrivate</i> did not unmarshal correctly
TPM_RC_SIZE	<i>inPrivate</i> missing, or <i>authPolicy</i> size for <i>inPublic</i> or is not valid
TPM_RC_SYMMETRIC	symmetric algorithm not provided when required
TPM_RC_TYPE	<i>parentHandle</i> is not a storage key, or the object to load is a storage key but its parameters do not match the parameters of the parent.
TPM_RC_VALUE	decryption failure

```

4  TPM_RC
5  TPM2_Load(
6      Load_In *in,           // IN: input parameter list
7      Load_Out *out         // OUT: output parameter list
8  )
9  {
10     TPM_RC          result = TPM_RC_SUCCESS;
11     TPMT_SENSITIVE sensitive;
12     TPMT_RH_HIERARCHY hierarchy;
13     OBJECT          *parentObject = NULL;
14     BOOL            skipChecks = FALSE;
15
16     // Input Validation
17     if(in->inPrivate.t.size == 0)
18         return TPM_RC_SIZE + RC_Load_inPrivate;
19
20     parentObject = ObjectGet(in->parentHandle);
21     // Is the object that is being used as the parent actually a parent.
22     if(!AreAttributesForParent(parentObject))
23         return TPM_RC_TYPE + RC_Load_parentHandle;
24
25     // If the parent is fixedTPM, then the attributes of the object
26     // are either "correct by construction" or were validated
27     // when the object was imported. If they pass the integrity
28     // check, then the values are valid
29     if(parentObject->publicArea.objectAttributes.fixedTPM)
30         skipChecks = TRUE;

```

```
31     else
32     {
33         // If parent doesn't have fixedTPM SET, then this can't have
34         // fixedTPM SET.
35         if(in->inPublic.t.publicArea.objectAttributes.fixedTPM == SET)
36             return TPM_RC_ATTRIBUTES + RC_Load_inPublic;
37
38         // Perform self check on input public area. A TPM_RC_SIZE, TPM_RC_SCHEME,
39         // TPM_RC_VALUE, TPM_RC_SYMMETRIC, TPM_RC_TYPE, TPM_RC_HASH,
40         // TPM_RC_ASYMMETRIC, TPM_RC_ATTRIBUTES or TPM_RC_KDF error may be returned
41         // at this point
42         result = PublicAttributesValidation(TRUE, in->parentHandle,
43                                           &in->inPublic.t.publicArea);
44         if(result != TPM_RC_SUCCESS)
45             return RcSafeAddToResult(result, RC_Load_inPublic);
46     }
47
48     // Compute the name of object
49     ObjectComputeName(&in->inPublic.t.publicArea, &out->name);
50
51     // Retrieve sensitive data. PrivateToSensitive() may return TPM_RC_INTEGRITY or
52     // TPM_RC_SENSITIVE
53     // errors may be returned at this point
54     result = PrivateToSensitive(&in->inPrivate, &out->name, in->parentHandle,
55                               in->inPublic.t.publicArea.nameAlg,
56                               &sensitive);
57     if(result != TPM_RC_SUCCESS)
58         return RcSafeAddToResult(result, RC_Load_inPrivate);
59
60     // Internal Data Update
61
62     // Get hierarchy of parent
63     hierarchy = ObjectGetHierarchy(in->parentHandle);
64
65     // Create internal object. A lot of different errors may be returned by this
66     // loading operation as it will do several validations, including the public
67     // binding check
68     result = ObjectLoad(hierarchy, &in->inPublic.t.publicArea, &sensitive,
69                       &out->name, in->parentHandle, skipChecks,
70                       &out->objectHandle);
71
72     if(result != TPM_RC_SUCCESS)
73         return result;
74
75     return TPM_RC_SUCCESS;
76 }
```

## 14.3 TPM2\_LoadExternal

### 14.3.1 General Description

This command is used to load an object that is not a Protected Object into the TPM. The command allows loading of a public area or both a public and sensitive area.

NOTE 1 Typical use for loading a public area is to allow the TPM to validate an asymmetric signature. Typical use for loading both a public and sensitive area is to allow the TPM to be used as a crypto accelerator.

Load of a public external object area allows the object be associated with a hierarchy so that the correct algorithms may be used when creating tickets. The *hierarchy* parameter provides this association. If the public and sensitive portions of the object are loaded, *hierarchy* is required to be TPM\_RH\_NULL.

NOTE 2 If both the public and private portions of an object are loaded, the object is not allowed to appear to be part of a hierarchy.

The object's TPMA\_OBJECT will be checked according to the rules defined in "TPMA\_OBJECT" in Part 2. In particular, *fixedTPM*, *fixedParent*, and *restricted* shall be CLEAR if *inPrivate* is not the Empty Buffer.

NOTE 3 The duplication status of a public key needs to be able to be the same as the full key which may be resident on a different TPM. If both the public and private parts of the key are loaded, then it is not possible for the key to be either *fixedTPM* or *fixedParent*, otherwise, its public area would not be available to load.

Objects loaded using this command will have a Name. The Name is the *nameAlg* of the object concatenated with the digest of the public area using the *nameAlg*. The Qualified Name for the object will be the same as its Name. The TPM will validate that the *authPolicy* is either the size of the digest produced by *nameAlg* or the Empty Buffer.

NOTE 4 If *nameAlg* is TPM\_ALG\_NULL, then the Name is the Empty Buffer. When the authorization value for an object with no Name is computed, no Name value is included in the HMAC. To ensure that these unnamed entities are not substituted, they should have an *authValue* that is statistically unique.

NOTE 5 The digest size for TPM\_ALG\_NULL is zero.

If the *nameAlg* is TPM\_ALG\_NULL, the TPM shall not verify the cryptographic binding between the public and sensitive areas, but the TPM will validate that the size of the key in the sensitive area is consistent with the size indicated in the public area. If it is not, the TPM shall return TPM\_RC\_KEY\_SIZE.

NOTE 6 For an ECC object, the TPM will verify that the public key is on the curve of the key before the public area is used.

If *nameAlg* is not TPM\_ALG\_NULL, then the same consistency checks between *inPublic* and *inPrivate* are made as for TPM2\_Load().

NOTE 7 Consistency checks are necessary because an object with a Name needs to have the public and sensitive portions cryptographically bound so that an attacker cannot mix public and sensitive areas.

The command returns a handle for the loaded object and the Name that the TPM computed for *inPublic.public* (that is, the TPMT\_PUBLIC structure in *inPublic*).

NOTE 8 The TPM-computed Name is provided as a convenience to the caller for those cases where the caller does not implement the hash algorithm specified in the *nameAlg* of the object.

The hierarchy parameter associates the external object with a hierarchy. External objects are flushed when their associated hierarchy is disabled.

If *hierarchy* is TPM\_RH\_NULL or *nameAlg* is TPM\_ALG\_NULL, a ticket produced using the object shall be a NULL Ticket.

EXAMPLE        If a key is loaded with hierarchy set to TPM\_RH\_NULL, then TPM2\_VerifySignature() will produce a NULL Ticket of the required type.

External objects are Temporary Objects. The saved external object contexts shall be invalidated at the next TPM Reset.

14.3.2 Command and Response

Table 23 — TPM2\_LoadExternal Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_LoadExternal
TPM2B_SENSITIVE	inPrivate	the sensitive portion of the object (optional)
TPM2B_PUBLIC+	inPublic	the public portion of the object
TPMI_RH_HIERARCHY+	hierarchy	hierarchy with which the object area is associated

Table 24 — TPM2\_LoadExternal Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM_HANDLE	objectHandle	handle for the loaded object
TPM2B_NAME	name	name of the loaded object

## 14.3.3 Detailed Actions

```

1 #include "InternalRoutines.h"
2 #include "LoadExternal_fp.h"
3 #include "Object_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	'fixedParent' and fixedTPM must be CLEAR on on an external key if both public and sensitive portions are loaded
TPM_RC_BINDING	the inPublic and inPrivate structures are not cryptographically bound.
TPM_RC_HASH	incorrect hash selection for signing key
TPM_RC_HIERARCHY	hierarchy is turned off, or only NULL hierarchy is allowed when loading public and private parts of an object
TPM_RC_KDF	incorrect KDF selection for decrypting keyedHash object
TPM_RC_KEY	the size of the object's unique field is not consistent with the indicated size in the object's parameters
TPM_RC_OBJECT_MEMORY	if there is no free slot for an object
TPM_RC_SCHEME	the signing scheme is not valid for the key
TPM_RC_SIZE	authPolicy is not zero and is not the size of a digest produced by the object's nameAlg TPM_RH_NULL hierarchy
TPM_RC_SYMMETRIC	symmetric algorithm not provided when required
TPM_RC_TYPE	inPublic and inPrivate are not the same type

```

4 TPM_RC
5 TPM2_LoadExternal(
6     LoadExternal_In *in,           // IN: input parameter list
7     LoadExternal_Out *out,        // OUT: output parameter list
8 )
9 {
10     TPM_RC result;
11     TPMT_SENSITIVE *sensitive;
12     BOOL skipChecks;
13
14     // Input Validation
15
16     // If the target hierarchy is turned off, the object can not be loaded.
17     if(!HierarchyIsEnabled(in->hierarchy))
18         return TPM_RC_HIERARCHY + RC_LoadExternal_hierarchy;
19
20     // the size of authPolicy is either 0 or the digest size of nameAlg
21     if(in->inPublic.t.publicArea.authPolicy.t.size != 0
22         && in->inPublic.t.publicArea.authPolicy.t.size !=
23         CryptGetHashDigestSize(in->inPublic.t.publicArea.nameAlg))
24         return TPM_RC_SIZE + RC_LoadExternal_inPublic;
25
26     // For loading an object with both public and sensitive
27     if(in->inPrivate.t.size != 0)
28     {
29         // An external object can only be loaded at TPM_RH_NULL hierarchy
30         if(in->hierarchy != TPM_RH_NULL)
31             return TPM_RC_HIERARCHY + RC_LoadExternal_hierarchy;
32         // An external object with a sensitive area must have fixedTPM == CLEAR
33         // fixedParent == CLEAR, and must have restrict CLEAR so that it does not

```



```
34     // appear to be a key that was created by this TPM.
35     if( in->inPublic.t.publicArea.objectAttributes.fixedTPM != CLEAR
36         || in->inPublic.t.publicArea.objectAttributes.fixedParent != CLEAR
37         || in->inPublic.t.publicArea.objectAttributes.restricted != CLEAR
38     )
39         return TPM_RC_ATTRIBUTES + RC_LoadExternal_inPublic;
40 }
41
42 // Validate the scheme parameters
43 result = SchemeChecks(TRUE, TPM_RH_NULL, &in->inPublic.t.publicArea);
44 if(result != TPM_RC_SUCCESS)
45     return RcSafeAddToResult(result, RC_LoadExternal_inPublic);
46
47
48 // Internal Data Update
49 // Need the name to compute the qualified name
50 ObjectComputeName(&in->inPublic.t.publicArea, &out->name);
51 skipChecks = (in->inPublic.t.publicArea.nameAlg == TPM_ALG_NULL);
52
53 // If a sensitive area was provided, load it
54 if(in->inPrivate.t.size != 0)
55     sensitive = &in->inPrivate.t.sensitiveArea;
56 else
57     sensitive = NULL;
58
59 // Create external object. A TPM_RC_BINDING, TPM_RC_KEY, TPM_RC_OBJECT_MEMORY
60 // or TPM_RC_TYPE error may be returned by ObjectLoad()
61 result = ObjectLoad(in->hierarchy, &in->inPublic.t.publicArea,
62                   sensitive, &out->name, TPM_RH_NULL, skipChecks,
63                   &out->objectHandle);
64 return result;
65 }
```

## 14.4 TPM2\_ReadPublic

### 14.4.1 General Description

This command allows access to the public area of a loaded object.

Use of the *objectHandle* does not require authorization.

NOTE                Since the caller is not likely to know the public area of the object associated with *objectHandle*, it would not be possible to include the Name associated with *objectHandle* in the *cpHash* computation.

If *objectHandle* references a sequence, the TPM shall return TPM\_RC\_SEQUENCE.

14.4.2 Command and Response

Table 25 — TPM2\_ReadPublic Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ReadPublic
TPMI_DH_OBJECT	objectHandle	TPM handle of an object Auth Index: None

Table 26 — TPM2\_ReadPublic Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PUBLIC	outPublic	structure containing the public area of an object
TPM2B_NAME	name	name of the object
TPM2B_NAME	qualifiedName	the Qualified Name of the object

## 14.4.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "ReadPublic_fp.h"

```

Error Returns	Meaning
TPM_RC_SEQUENCE	can not read the public area of a sequence object

```

3  TPM_RC
4  TPM2_ReadPublic(
5      ReadPublic_In      *in,          // IN: input parameter list
6      ReadPublic_Out     *out         // OUT: output parameter list
7  )
8  {
9      OBJECT              *object;
10
11     // Input Validation
12
13     // Get loaded object pointer
14     object = ObjectGet(in->objectHandle);
15
16     // Can not read public area of a sequence object
17     if(ObjectIsSequence(object))
18         return TPM_RC_SEQUENCE;
19
20
21     // Command Output
22
23     // Compute size of public area in canonical form
24     out->outPublic.t.size = TPMT_PUBLIC_Marshal(&object->publicArea, NULL, NULL);
25
26     // Copy public area to output
27     out->outPublic.t.publicArea = object->publicArea;
28
29     // Copy name to output
30     out->name.t.size = ObjectGetName(in->objectHandle, out->name.t.name);
31
32     // Copy qualified name to output
33     ObjectGetQualifiedName(in->objectHandle, &out->qualifiedName);
34
35     return TPM_RC_SUCCESS;
36 }

```

## 14.5 TPM2\_ActivateCredential

### 14.5.1 General Description

This command enables the association of a credential with an object in a way that ensures that the TPM has validated the parameters of the credentialed object.

If both the public and private portions of *activateHandle* and *keyHandle* are not loaded, then the TPM shall return TPM\_RC\_AUTH\_UNAVAILABLE.

If *keyHandle* is not a Storage Key, then the TPM shall return TPM\_RC\_TYPE.

Authorization for *activateHandle* requires the ADMIN role.

The key associated with *keyHandle* is used to recover a symmetric key and an HMAC key from *secret*.

The HMAC is used to validate that the *credentialBlob* is associated with *activateHandle* and that the data in *credentialBlob* has not been modified.

If the integrity checks succeed, *credentialBlob* is decrypted and returned as *certInfo*.

## 14.5.2 Command and Response

Table 27 — TPM2\_ActivateCredential Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ActivateCredential
TPMI_DH_OBJECT	@activateHandle	handle of the object associated with certificate in <i>credentialBlob</i> Auth Index: 1 Auth Role: ADMIN
TPMI_DH_OBJECT	@keyHandle	loaded key used to decrypt the TPMS_SENSITIVE in <i>credentialBlob</i> Auth Index: 2 Auth Role: USER
TPM2B_ID_OBJECT	credentialBlob	the credential
TPM2B_ENCRYPTED_SECRET	secret	<i>keyHandle</i> algorithm-dependent data that wraps the key that encrypts <i>credentialBlob</i>

Table 28 — TPM2\_ActivateCredential Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DIGEST	certInfo	the decrypted certificate information the data should be no larger than the size of the digest of the <i>nameAlg</i> associated with <i>keyHandle</i>

## 14.5.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "ActivateCredential_fp.h"
3  #include "Object_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	<i>keyHandle</i> does not reference a decryption key
TPM_RC_ECC_POINT	<i>secret</i> is invalid (when <i>keyHandle</i> is an ECC key)
TPM_RC_INSUFFICIENT	<i>secret</i> is invalid (when <i>keyHandle</i> is an ECC key)
TPM_RC_INTEGRITY	<i>credentialBlob</i> fails integrity test
TPM_RC_NO_RESULT	<i>secret</i> is invalid (when <i>keyHandle</i> is an ECC key)
TPM_RC_SIZE	<i>secret</i> size is invalid or the <i>credentialBlob</i> does not unmarshal correctly
TPM_RC_TYPE	<i>keyHandle</i> does not reference an asymmetric key.
TPM_RC_VALUE	<i>secret</i> is invalid (when <i>keyHandle</i> is an RSA key)

```

4  TPM_RC
5  TPM2_ActivateCredential(
6      ActivateCredential_In      *in,           // IN: input parameter list
7      ActivateCredential_Out     *out          // OUT: output parameter list
8  )
9  {
10     TPM_RC          result = TPM_RC_SUCCESS;
11     OBJECT          *object; // decrypt key
12     OBJECT          *activateObject; // key associated with
13     // credential
14     TPM2B_DATA      data; // credential data
15
16     // Input Validation
17
18     // Get decrypt key pointer
19     object = ObjectGet(in->keyHandle);
20
21     // Get certificated object pointer
22     activateObject = ObjectGet(in->activateHandle);
23
24
25     // input decrypt key must be an asymmetric, restricted decryption key
26     if( !CryptIsAsymAlgorithm(object->publicArea.type)
27         || object->publicArea.objectAttributes.decrypt == CLEAR
28         || object->publicArea.objectAttributes.restricted == CLEAR)
29         return TPM_RC_TYPE + RC_ActivateCredential_keyHandle;
30
31     // Command output
32
33     // Decrypt input credential data via asymmetric decryption. A
34     // TPM_RC_VALUE, TPM_RC_KEY or unmarshal errors may be returned at this
35     // point
36     result = CryptSecretDecrypt(in->keyHandle, NULL,
37                                "IDENTITY", &in->secret, &data);
38     if(result != TPM_RC_SUCCESS)
39     {
40         if(result == TPM_RC_KEY)
41             return TPM_RC_FAILURE;
42         return RcSafeAddToResult(result, RC_ActivateCredential_secret);

```

```
43     }
44
45     // Retrieve secret data. A TPM_RC_INTEGRITY error or unmarshal
46     // errors may be returned at this point
47     result = CredentialToSecret(&in->credentialBlob,
48                               &activateObject->name,
49                               (TPM2B_SEED *) &data,
50                               in->keyHandle,
51                               &out->certInfo);
52     if(result != TPM_RC_SUCCESS)
53         return RcSafeAddToResult(result,RC_ActivateCredential_credentialBlob);
54
55     return TPM_RC_SUCCESS;
56 }
```



## 14.6 TPM2\_MakeCredential

### 14.6.1 General Description

This command allows the TPM to perform the actions required of a Certificate Authority (CA) in creating a TPM2B\_ID\_OBJECT containing an activation credential.

The TPM will produce a TPM\_ID\_OBJECT according to the methods in “Credential Protection” in Part 1.

The loaded public area referenced by *handle* is required to be the public area of a Storage key, otherwise, the credential cannot be properly sealed.

## 14.6.2 Command and Response

Table 29 — TPM2\_MakeCredential Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_MakeCredential
TPMI_DH_OBJECT	handle	loaded public area, used to encrypt the sensitive area containing the credential key Auth Index: None
TPM2B_DIGEST	credential	the credential information
TPM2B_NAME	objectName	Name of the object to which the credential applies

Table 30 — TPM2\_MakeCredential Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ID_OBJECT	credentialBlob	the credential
TPM2B_ENCRYPTED_SECRET	secret	<i>handle</i> algorithm-dependent data that wraps the key that encrypts <i>credentialBlob</i>

## 14.6.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "MakeCredential_fp.h"
3  #include "Object_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_KEY	<i>handle</i> referenced an ECC key that has a unique field that is not a point on the curve of the key
TPM_RC_SIZE	<i>credential</i> is larger than the digest size of Name algorithm of <i>handle</i>
TPM_RC_TYPE	<i>handle</i> does not reference an asymmetric decryption key

```

4  TPM_RC
5  TPM2_MakeCredential(
6      MakeCredential_In      *in,          // IN: input parameter list
7      MakeCredential_Out     *out         // OUT: output parameter list
8  )
9  {
10     TPM_RC                  result = TPM_RC_SUCCESS;
11
12     OBJECT                  *object;
13     TPM2B_DATA              data;
14
15     // Input Validation
16
17     // Get object pointer
18     object = ObjectGet(in->handle);
19
20     // input key must be an asymmetric, restricted decryption key
21     // NOTE: Needs to be restricted to have a symmetric value.
22     if( !CryptIsAsymAlgorithm(object->publicArea.type)
23         || object->publicArea.objectAttributes.decrypt == CLEAR
24         || object->publicArea.objectAttributes.restricted == CLEAR
25     )
26         return TPM_RC_TYPE + RC_MakeCredential_handle;
27
28     // The credential information may not be larger than the digest size used for
29     // the Name of the key associated with handle.
30     if(in->credential.t.size > CryptGetHashDigestSize(object->publicArea.nameAlg))
31         return TPM_RC_SIZE + RC_MakeCredential_credential;
32
33     // Command Output
34
35     // Make encrypt key and its associated secret structure.
36     // Even though CrypeSecretEncrypt() may return
37     out->secret.t.size = sizeof(out->secret.t.secret);
38     result = CryptSecretEncrypt(in->handle, "IDENTITY", &data, &out->secret);
39     if(result != TPM_RC_SUCCESS)
40         return result;
41
42     // Prepare output credential data from secret
43     SecretToCredential(&in->credential, &in->objectName, (TPM2B_SEED *) &data,
44                     in->handle, &out->credentialBlob);
45
46     return TPM_RC_SUCCESS;
47 }

```

## 14.7 TPM2\_Unseal

### 14.7.1 General Description

This command returns the data in a loaded Sealed Data Object.

NOTE           A random, TPM-generated, Sealed Data Object may be created by the TPM with TPM2\_Create() or TPM2\_CreatePrimary() using the template for a Sealed Data Object. A Sealed Data Object is more likely to be created externally and imported (TPM2\_Import()) so that the data is not created by the TPM.

The returned value may be encrypted using authorization session encryption.

If either *restricted*, *decrypt*, or *sign* is SET in the attributes of *itemHandle*, then the TPM shall return TPM\_RC\_ATTRIBUTES. If the *type* of *itemHandle* is not TPM\_ALG\_KEYEDHASH, then the TPM shall return TPM\_RC\_TYPE.

14.7.2 Command and Response

Table 31 — TPM2\_Unseal Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	Tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Unseal
TPMI_DH_OBJECT	@itemHandle	handle of a loaded data object Auth Index: 1 Auth Role: USER

Table 32 — TPM2\_Unseal Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_SENSITIVE_DATA	outData	unsealed data Size of <i>outData</i> is limited to be no more than 128 octets.

## 14.7.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Unseal_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	<i>itemHandle</i> has wrong attributes
TPM_RC_TYPE	<i>itemHandle</i> is not a KEYEDHASH data object

```

3  TPM_RC
4  TPM2_Unseal(Unseal_In *in, Unseal_Out *out)
5  {
6      OBJECT                *object;
7
8      // Input Validation
9
10     // Get pointer to loaded object
11     object = ObjectGet(in->itemHandle);
12
13     // Input handle must be a data object
14     if(object->publicArea.type != TPM_ALG_KEYEDHASH)
15         return TPM_RC_TYPE + RC_Unseal_itemHandle;
16     if( object->publicArea.objectAttributes.decrypt == SET
17         || object->publicArea.objectAttributes.sign == SET
18         || object->publicArea.objectAttributes.restricted == SET)
19         return TPM_RC_ATTRIBUTES + RC_Unseal_itemHandle;
20
21     // Command Output
22
23     // Copy data
24     MemoryCopy2B(&out->outData.b, &object->sensitive.sensitive.bits.b);
25
26     return TPM_RC_SUCCESS;
27 }

```

## 14.8 TPM2\_ObjectChangeAuth

### 14.8.1 General Description

This command is used to change the authorization secret for a TPM-resident object.

If successful, the authorization secret (*authValue*) of the TPM-resident object associated with *objectHandle* returns a new private area with the new authorization value. This command does not change the authorization of the TPM-resident object on which it operates.

NOTE 1            The returned *outPrivate* will need to be loaded before the new authorization will apply.

NOTE 2            The TPM-resident object may be persistent and changing the authorization value of the persistent object could prevent other users from accessing the object. This is why this command does not change the TPM-resident object.

EXAMPLE          If a persistent key is being used as a Storage Root Key and the authorization of the key is a well-known value so that the key can be used generally, then changing the authorization value in the persistent key would deny access to other users.

This command may not be used to change the authorization value for an NV Index or a Primary Object.

NOTE 3            If an NV Index is to have a new authorization, it is done with TPM2\_NV\_ChangeAuth().

NOTE 4            If a Primary Object is to have a new authorization, it needs to be recreated (TPM2\_CreatePrimary()).

## 14.8.2 Command and Response

Table 33 — TPM2\_ObjectChangeAuth Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ObjectChangeAuth
TPMI_DH_OBJECT	@objectHandle	handle of the object Auth Index: 1 Auth Role: ADMIN
TPMI_DH_OBJECT	parentHandle	handle of the parent Auth Index: None
TPM2B_AUTH	newAuth	new authorization secret

Table 34 — TPM2\_ObjectChangeAuth Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PRIVATE	outPrivate	private area containing the new authorization value



## 14.8.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "ObjectChangeAuth_fp.h"
3  #include "Object_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_SIZE	<i>newAuth</i> is larger than the size of the digest of the Name algorithm of <i>objectHandle</i>
TPM_RC_TYPE	the key referenced by <i>parentHandle</i> is not the parent of the object referenced by <i>objectHandle</i> ; or <i>objectHandle</i> is a sequence object.

```

4  TPM_RC
5  TPM2_ObjectChangeAuth(
6      ObjectChangeAuth_In      *in,          // IN: input parameter list
7      ObjectChangeAuth_Out     *out         // OUT: output parameter list
8  )
9  {
10     TPMT_SENSITIVE            sensitive;
11
12     OBJECT                    *object;
13     TPM2B_NAME                objectQN, QNCompare;
14     TPM2B_NAME                parentQN;
15
16     // Input Validation
17
18     // Get object pointer
19     object = ObjectGet(in->objectHandle);
20
21     // Can not change auth on sequence object
22     if(ObjectIsSequence(object))
23         return TPM_RC_TYPE + RC_ObjectChangeAuth_objectHandle;
24
25     // Make sure that the auth value is consistent with the nameAlg
26     if( MemoryRemoveTrailingZeros(&in->newAuth)
27         > CryptGetHashDigestSize(object->publicArea.nameAlg))
28         return TPM_RC_SIZE + RC_ObjectChangeAuth_newAuth;
29
30
31     // Check parent for object
32     // parent handle must be the parent of object handle. In this
33     // implementation we verify this by checking the QN of object. Other
34     // implementation may choose different method to verify this attribute.
35     ObjectGetQualifiedName(in->parentHandle, &parentQN);
36     ObjectComputeQualifiedName(&parentQN, object->publicArea.nameAlg,
37                               &object->name, &QNCompare);
38
39     ObjectGetQualifiedName(in->objectHandle, &objectQN);
40     if(!Memory2BEqual(&objectQN.b, &QNCompare.b))
41         return TPM_RC_TYPE + RC_ObjectChangeAuth_parentHandle;
42
43     // Command Output
44
45     // Copy internal sensitive area
46     sensitive = object->sensitive;
47     // Copy authValue
48     sensitive.authValue = in->newAuth;
49
50     // Prepare output private data from sensitive
51     SensitiveToPrivate(&sensitive, &object->name, in->parentHandle,
52                       object->publicArea.nameAlg,

```

```
53         &out->outPrivate);  
54  
55     return TPM_RC_SUCCESS;  
56 }
```

## 15 Duplication Commands

### 15.1 TPM2\_Duplicate

#### 15.1.1 General Description

This command duplicates a loaded object so that it may be used in a different hierarchy. The new parent key for the duplicate may be on the same or different TPM or TPM\_RH\_NULL. Only the public area of *newParentHandle* is required to be loaded.

NOTE 1            Since the new parent may only be extant on a different TPM, it is likely that the new parent's sensitive area could not be loaded in the TPM from which *objectHandle* is being duplicated.

If *encryptedDuplication* is SET in the object being duplicated, then the TPM shall return TPM\_RC\_SYMMETRIC if *symmetricAlg* is TPM\_RH\_NULL or TPM\_RC\_HIERARCHY if *newParentHandle* is TPM\_RH\_NULL.

The authorization for this command shall be with a policy session.

If *fixedParent* of *objectHandle*→*attributes* is SET, the TPM shall return TPM\_RC\_ATTRIBUTES. If *objectHandle*→*nameAlg* is TPM\_ALG\_NULL, the TPM shall return TPM\_RC\_TYPE.

The *policySession*→*commandCode* parameter in the policy session is required to be TPM\_CC\_Duplicate to indicate that authorization for duplication has been provided.

If TPM2\_PolicyCpHash() has been executed as part of the policy, the *policySession*→*cpHash* is compared to the cpHash of the command. If TPM2\_PolicyDuplicationSelect() has been executed as part of the policy, the *policySession*→*nameHash* is compared to

$$H_{policyAlg}(objectHandle \rightarrow Name || newParentHandle \rightarrow Name) \quad (2)$$

If the compared hashes are not the same, then the TPM shall return TPM\_RC\_POLICY\_FAIL.

NOTE 2            A duplication policy is not required to have either TPM2\_PolicyDuplicationSelect() or TPM2\_PolicyCpHash() as part of the policy. If neither is present, then the duplication policy may be satisfied with a policy that only contains TPM2\_PolicyCommandCode(*code* = TPM\_CC\_Duplicate).

The TPM shall follow the process of encryption defined in the “Duplication” subclause of “Protected Storage Hierarchy” in Part 1 of this specification.

## 15.1.2 Command and Response

Table 35 — TPM2\_Duplicate Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Duplicate
TPMI_DH_OBJECT	@objectHandle	loaded object to duplicate Auth Index: 1 Auth Role: DUP
TPMI_DH_OBJECT+	newParentHandle	shall reference the public area of an asymmetric key Auth Index: None
TPM2B_DATA	encryptionKeyIn	optional symmetric encryption key The size for this key is set to zero when the TPM is to generate the key. This parameter may be encrypted.
TPMT_SYM_DEF_OBJECT+	symmetricAlg	definition for the symmetric algorithm to be used for the inner wrapper may be TPM_ALG_NULL if no inner wrapper is applied

Table 36 — TPM2\_Duplicate Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DATA	encryptionKeyOut	If the caller provided an encryption key or if <i>symmetricAlg</i> was TPM_ALG_NULL, then this will be the Empty Buffer; otherwise, it shall contain the TPM-generated, symmetric encryption key for the inner wrapper.
TPM2B_PRIVATE	duplicate	private area that may be encrypted by <i>encryptionKeyIn</i> ; and may be doubly encrypted
TPM2B_ENCRYPTED_SECRET	outSymSeed	seed protected by the asymmetric algorithms of new parent (NP)

## 15.1.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Duplicate_fp.h"
3  #include "Object_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	key to duplicate has <i>fixedParent</i> SET
TPM_RC_HIERARCHY	<i>encryptedDuplication</i> is SET and <i>newParentHandle</i> specifies Null Hierarchy
TPM_RC_KEY	<i>newParentHandle</i> references invalid ECC key (public point not on the curve)
TPM_RC_SIZE	input encryption key size does not match the size specified in symmetric algorithm
TPM_RC_SYMMETRIC	<i>encryptedDuplication</i> is SET but no symmetric algorithm is provided
TPM_RC_TYPE	<i>newParentHandle</i> is neither a storage key nor TPM_RH_NULL; or the object has a NULL <i>nameAlg</i>

```

4  TPM_RC
5  TPM2_Duplicate(
6      Duplicate_In    *in,           // IN: input parameter list
7      Duplicate_Out   *out,         // OUT: output parameter list
8  )
9  {
10     TPM_RC          result = TPM_RC_SUCCESS;
11     TPMT_SENSITIVE sensitive;
12
13     UINT16          innerKeySize = 0; // encrypt key size for inner wrap
14
15     OBJECT          *object;
16     TPM2B_DATA      data;
17
18     // Input Validation
19
20     // Get duplicate object pointer
21     object = ObjectGet(in->objectHandle);
22
23     // duplicate key must have fixParent bit CLEAR.
24     if(object->publicArea.objectAttributes.fixedParent == SET)
25         return TPM_RC_ATTRIBUTES + RC_Duplicate_objectHandle;
26
27     // Do not duplicate object with NULL nameAlg
28     if(object->publicArea.nameAlg == TPM_ALG_NULL)
29         return TPM_RC_TYPE + RC_Duplicate_objectHandle;
30
31     // new parent key must be a storage object or TPM_RH_NULL
32     if(in->newParentHandle != TPM_RH_NULL
33         && !ObjectIsStorage(in->newParentHandle))
34         return TPM_RC_TYPE + RC_Duplicate_newParentHandle;
35
36     // If the duplicates object has encryptedDuplication SET, then there must be
37     // an inner wrapper and the new parent may not be TPM_RH_NULL
38     if(object->publicArea.objectAttributes.encryptedDuplication == SET)
39     {
40         if(in->symmetricAlg.algorithm == TPM_ALG_NULL)
41             return TPM_RC_SYMMETRIC + RC_Duplicate_symmetricAlg;
42         if(in->newParentHandle == TPM_RH_NULL)
43             return TPM_RC_HIERARCHY + RC_Duplicate_newParentHandle;
44     }

```

```

45
46     if(in->symmetricAlg.algorithm == TPM_ALG_NULL)
47     {
48         // if algorithm is TPM_ALG_NULL, input key size must be 0
49         if(in->encryptionKeyIn.t.size != 0)
50             return TPM_RC_SIZE + RC_Duplicate_encryptionKeyIn;
51     }
52     else
53     {
54         // Get inner wrap key size
55         innerKeySize = in->symmetricAlg.keyBits.sym;
56
57         // If provided the input symmetric key must match the size of the algorithm
58         if(in->encryptionKeyIn.t.size != 0
59            && in->encryptionKeyIn.t.size != (innerKeySize + 7) / 8)
60             return TPM_RC_SIZE + RC_Duplicate_encryptionKeyIn;
61     }
62
63     // Command Output
64
65     if(in->newParentHandle != TPM_RH_NULL)
66     {
67
68         // Make encrypt key and its associated secret structure. A TPM_RC_KEY
69         // error may be returned at this point
70         out->outSymSeed.t.size = sizeof(out->outSymSeed.t.secret);
71         result = CryptSecretEncrypt(in->newParentHandle,
72                                   "DUPLICATE", &data, &out->outSymSeed);
73         pAssert(result != TPM_RC_VALUE);
74         if(result != TPM_RC_SUCCESS)
75             return result;
76     }
77     else
78     {
79         // Do not apply outer wrapper
80         data.t.size = 0;
81         out->outSymSeed.t.size = 0;
82     }
83
84     // Copy sensitive area
85     sensitive = object->sensitive;
86
87     // Prepare output private data from sensitive
88     SensitiveToDuplicate(&sensitive, &object->name, in->newParentHandle,
89                        object->publicArea.nameAlg, (TPM2B_SEED *) &data,
90                        &in->symmetricAlg, &in->encryptionKeyIn,
91                        &out->duplicate);
92
93     out->encryptionKeyOut = in->encryptionKeyIn;
94
95     return TPM_RC_SUCCESS;
96 }

```

## 15.2 TPM2\_Rewrap

### 15.2.1 General Description

This command allows the TPM to serve in the role as a Duplication Authority. If proper authorization for use of the *oldParent* is provided, then a symmetric key is recovered from *inSymKey* and used to integrity check and decrypt *inDuplicate*. A new protection seed value is generated according to the methods appropriate for *newParent* and the blob is re-encrypted and a new integrity value is computed. The re-encrypted blob is returned in *outDuplicate* and the symmetric key returned in *outSymKey*.

In the rewrap process, L is “DUPLICATE” (see “Terms and Definitions” in Part 1).

If *inSymSeed* has a zero length, then *oldParent* is required to be TPM\_RH\_NULL and no decryption of *inDuplicate* takes place.

If *newParent* is TPM\_RH\_NULL, then no encryption is performed on *outDuplicate* and *outSymSeed* will have a zero length.

## 15.2.2 Command and Response

Table 37 — TPM2\_Rewrap Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Rewrap
TPMI_DH_OBJECT+	@oldParent	parent of object Auth Index: 1 Auth Role: User
TPMI_DH_OBJECT+	newParent	new parent of the object Auth Index: None
TPM2B_PRIVATE	inDuplicate	an object encrypted using symmetric key derived from <i>inSymSeed</i>
TPM2B_NAME	name	the Name of the object being rewrapped
TPM2B_ENCRYPTED_SECRET	inSymSeed	seed for symmetric key needs <i>oldParent</i> private key to recover the seed and generate the symmetric key

Table 38 — TPM2\_Rewrap Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PRIVATE	outDuplicate	an object encrypted using symmetric key derived from <i>outSymSeed</i>
TPM2B_ENCRYPTED_SECRET	outSymSeed	seed for a symmetric key protected by <i>newParent</i> asymmetric key



## 15.2.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Rewrap_fp.h"
3  #include "Object_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	<i>newParent</i> is not a decryption key
TPM_RC_HANDLE	<i>oldParent</i> does not consistent with <i>inSymSeed</i>
TPM_RC_INTEGRITY	the integrity check of <i>inDuplicate</i> failed
TPM_RC_KEY	for an ECC key, the public key is not on the curve of the curve ID
TPM_RC_KEY_SIZE	the decrypted input symmetric key size does not matches the symmetric algorithm key size of <i>oldParent</i>
TPM_RC_TYPE	<i>oldParent</i> is not a storage key, or ' <i>newParent</i> ' is not a storage key
TPM_RC_VALUE	for an ' <i>oldParent</i> ', RSA key, the data to be decrypted is greater than the public exponent
Unmarshal errors	errors during unmarshaling the input encrypted buffer to a ECC public key, or unmarshal the private buffer to sensitive

```

4  TPM_RC
5  TPM2_Rewrap(
6      Rewrap_In          *in,          // IN: input parameter list
7      Rewrap_Out        *out         // OUT: output parameter list
8  )
9  {
10     TPM_RC              result = TPM_RC_SUCCESS;
11     OBJECT              *oldParent;
12     TPM2B_DATA          data;        // symmetric key
13     UINT16              hashCode = 0;
14     TPM2B_PRIVATE       privateBlob; // A temporary private blob
15                                     // to transit between old
16                                     // and new wrappers
17
18     // Input Validation
19
20     if((in->inSymSeed.t.size == 0 && in->oldParent != TPM_RH_NULL)
21        || (in->inSymSeed.t.size != 0 && in->oldParent == TPM_RH_NULL))
22         return TPM_RC_HANDLE + RC_Rewrap_oldParent;
23
24     if(in->oldParent != TPM_RH_NULL)
25     {
26         // Get old parent pointer
27         oldParent = ObjectGet(in->oldParent);
28
29         // old parent key must be a storage object
30         if(!ObjectIsStorage(in->oldParent))
31             return TPM_RC_TYPE + RC_Rewrap_oldParent;
32
33         // Decrypt input secret data via asymmetric decryption. A
34         // TPM_RC_VALUE, TPM_RC_KEY or unmarshal errors may be returned at this
35         // point
36         result = CryptSecretDecrypt(in->oldParent, NULL,
37                                    "DUPLICATE", &in->inSymSeed, &data);
38         if(result != TPM_RC_SUCCESS)
39             return TPM_RC_VALUE + RC_Rewrap_inSymSeed;

```

```

40
41 // Unwrap Outer
42 result = UnwrapOuter(in->oldParent, &in->name,
43                     oldParent->publicArea.nameAlg, (TPM2B_SEED *) &data,
44                     FALSE,
45                     in->inDuplicate.t.size, in->inDuplicate.t.buffer);
46 if(result != TPM_RC_SUCCESS)
47     return RcSafeAddToResult(result, RC_Rewrap_inDuplicate);
48
49 // Copy unwrapped data to temporary variable, remove the integrity field
50 hashSize = sizeof(UINT16) +
51           CryptGetHashDigestSize(oldParent->publicArea.nameAlg);
52 privateBlob.t.size = in->inDuplicate.t.size - hashSize;
53 MemoryCopy(privateBlob.t.buffer, in->inDuplicate.t.buffer + hashSize,
54            privateBlob.t.size);
55 }
56 else
57 {
58     // No outer wrap from input blob. Direct copy.
59     privateBlob = in->inDuplicate;
60 }
61
62 if(in->newParent != TPM_RH_NULL)
63 {
64     OBJECT *newParent;
65     newParent = ObjectGet(in->newParent);
66
67     // New parent must be a storage object
68     if(!ObjectIsStorage(in->newParent))
69         return TPM_RC_TYPE + RC_Rewrap_newParent;
70
71     // Make new encrypt key and its associated secret structure. A
72     // TPM_RC_VALUE error may be returned at this point if RSA algorithm is
73     // enabled in TPM
74     out->outSymSeed.t.size = sizeof(out->outSymSeed.t.secret);
75     result = CryptSecretEncrypt(in->newParent,
76                                "DUPLICATE", &data, &out->outSymSeed);
77     if(result != TPM_RC_SUCCESS) return result;
78
79 // Command output
80 // Copy temporary variable to output, reserve the space for integrity
81 hashSize = sizeof(UINT16) +
82           CryptGetHashDigestSize(newParent->publicArea.nameAlg);
83 out->outDuplicate.t.size = privateBlob.t.size;
84 MemoryCopy(out->outDuplicate.t.buffer + hashSize, privateBlob.t.buffer,
85            privateBlob.t.size);
86
87 // Produce outer wrapper for output
88 out->outDuplicate.t.size = ProduceOuterWrap(in->newParent, &in->name,
89                                           newParent->publicArea.nameAlg,
90                                           (TPM2B_SEED *) &data,
91                                           FALSE,
92                                           out->outDuplicate.t.size,
93                                           out->outDuplicate.t.buffer);
94 }
95
96 else // New parent is a null key so there is no seed
97 {
98     out->outSymSeed.t.size = 0;
99
100 // Copy privateBlob directly
101 out->outDuplicate = privateBlob;
102 }
103

```

```
104     return TPM_RC_SUCCESS;  
105 }
```

## 15.3 TPM2\_Import

### 15.3.1 General Description

This command allows an object to be encrypted using the symmetric encryption values of a Storage Key. After encryption, the object may be loaded and used in the new hierarchy. The imported object (*duplicate*) may be singly encrypted, multiply encrypted, or unencrypted.

If *fixedTPM* or *fixedParent* is SET in *objectPublic*, the TPM shall return TPM\_RC\_ATTRIBUTES.

If *encryptedDuplication* is SET in the object referenced by *parentHandle*, then *encryptedDuplication* shall be set in *objectPublic* (TPM\_RC\_ATTRIBUTES).

Recovery of the sensitive data of the object occurs in the TPM in a three-step process in the following order:

- If present, the outer layer of symmetric encryption is removed. If *inSymSeed* has a non-zero size, the asymmetric parameters and private key of *parentHandle* are used to recover the seed used in the creation of the HMAC key and encryption keys used to protect the duplication blob. When recovering the seed, *L* is “DUPLICATE”.

NOTE 1 If the *encryptedDuplication* attribute of the object is SET, the TPM shall return TPM\_RC\_ATTRIBUTES if *inSymSeed* is an empty buffer.

- If present, the inner layer of symmetric encryption is removed. If *encryptionKey* and *symmetricAlg* are provided, they are used to decrypt duplication.
- If present, the integrity value of the blob is checked. The presence of the integrity value is indicated by a non-zero value for *duplicate.data.integrity.size*. The integrity of the private area is validated using the Name of *objectPublic* in the integrity HMAC computation. If either the outer layer or inner layer of encryption is performed, then the integrity value shall be present.

If the inner or outer wrapper is present, then a valid integrity value shall be present or the TPM shall return TPM\_RC\_INTEGRITY.

NOTE 2 It is not necessary to validate that the sensitive area data is cryptographically bound to the public area other than that the Name of the public area is included in the HMAC. However, if the binding is not validated by this command, the binding must be checked each time the object is loaded. For an object that is imported under a parent with *fixedTPM* SET, binding need only be checked at import. If the parent has *fixedTPM* CLEAR, then the binding needs to be checked each time the object is loaded, or before the TPM performs an operation for which the binding affects the outcome of the operation (for example, TPM2\_PolicySigned() or TPM2\_Certify()).

After decryption and integrity checks, the TPM will create a new symmetrically encrypted private area using the encryption key of the parent.

After *inPrivate.buffer* is decrypted using the symmetric key of the parent, the integrity value shall be checked before the sensitive area is used, or unmarshaled.

NOTE 3 Checking the integrity before the data is used prevents attacks on the sensitive area by fuzzing the data and looking at the differences in the response codes.

NOTE 4 The symmetric re-encryption is the normal integrity generation and symmetric encryption applied to a child object.

## 15.3.2 Command and Response

Table 39 — TPM2\_Import Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Import
TPMI_DH_OBJECT	@parentHandle	the handle of the new parent for the object Auth Index: 1 Auth Role: USER
TPM2B_DATA	encryptionKey	the optional symmetric encryption key used as the inner wrapper for <i>duplicate</i> If <i>symmetricAlg</i> is TPM_ALG_NULL, then this parameter shall be the Empty Buffer.
TPM2B_PUBLIC	objectPublic	the public area of the object to be imported This is provided so that the integrity value for <i>duplicate</i> and the object attributes can be checked. NOTE Even if the integrity value of the object is not checked on input, the object Name is required to create the integrity value for the imported object.
TPM2B_PRIVATE	duplicate	the symmetrically encrypted duplicate object that may contain an inner symmetric wrapper
TPM2B_ENCRYPTED_SECRET	inSymSeed	symmetric key used to encrypt <i>duplicate</i> <i>inSymSeed</i> is encrypted/encoded using the algorithms of <i>newParent</i> .
TPMT_SYM_DEF_OBJECT+	symmetricAlg	definition for the symmetric algorithm to use for the inner wrapper If this algorithm is TPM_ALG_NULL, no inner wrapper is present and <i>encryptionKey</i> shall be the Empty Buffer.

Table 40 — TPM2\_Import Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PRIVATE	outPrivate	the sensitive area encrypted with the symmetric key of <i>parentHandle</i>

## 15.3.3 Detailed Actions

```

1 #include "InternalRoutines.h"
2 #include "Import_fp.h"
3 #include "Object_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_ASYMMETRIC	non-duplicable storage key represented by <i>objectPublic</i> and its parent referenced by <i>parentHandle</i> have different public params
TPM_RC_ATTRIBUTES	attributes <i>FixedTPM</i> and <i>fixedParent</i> of <i>objectPublic</i> are not both CLEAR; or <i>inSymSeed</i> is nonempty and <i>parentHandle</i> does not reference a decryption key; or <i>objectPublic</i> and <i>parentHandle</i> have incompatible or inconsistent attributes
TPM_RC_BINDING	<i>duplicate</i> and <i>objectPublic</i> are not cryptographically bound
TPM_RC_ECC_POINT	<i>inSymSeed</i> is nonempty and ECC point in <i>inSymSeed</i> is not on the curve
TPM_RC_HASH	non-duplicable storage key represented by <i>objectPublic</i> and its parent referenced by <i>parentHandle</i> have different name algorithm
TPM_RC_INSUFFICIENT	<i>inSymSeed</i> is nonempty and failed to retrieve ECC point from the secret; or unmarshaling sensitive value from <i>duplicate</i> failed the result of <i>inSymSeed</i> decryption
TPM_RC_INTEGRITY	<i>duplicate</i> integrity is broken
TPM_RC_KDF	<i>objectPublic</i> representing decrypting keyed hash object specifies invalid KDF
TPM_RC_KEY	inconsistent parameters of <i>objectPublic</i> ; or <i>inSymSeed</i> is nonempty and <i>parentHandle</i> does not reference a key of supported type; or invalid key size in <i>objectPublic</i> representing an asymmetric key
TPM_RC_NO_RESULT	<i>inSymSeed</i> is nonempty and multiplication resulted in ECC point at infinity
TPM_RC_OBJECT_MEMORY	no available object slot
TPM_RC_SCHEME	inconsistent attributes <i>decrypt</i> , <i>sign</i> , <i>restricted</i> and key's scheme ID in <i>objectPublic</i> ; or hash algorithm is inconsistent with the scheme ID for keyed hash object
TPM_RC_SIZE	<i>authPolicy</i> size does not match digest size of the name algorithm in <i>objectPublic</i> ; or <i>symmetricAlg</i> and <i>encryptionKey</i> have different sizes; or <i>inSymSeed</i> is nonempty and it is not of the same size as RSA key referenced by <i>parentHandle</i> ; or unmarshaling sensitive value from <i>duplicate</i> failed
TPM_RC_SYMMETRIC	<i>objectPublic</i> is either a storage key with no symmetric algorithm or a non-storage key with symmetric algorithm different from TPM_ALG_NULL
TPM_RC_TYPE	unsupported type of <i>objectPublic</i> ; or non-duplicable storage key represented by <i>objectPublic</i> and its parent referenced by <i>parentHandle</i> are of different types; or <i>parentHandle</i> is not a storage key; or only the public portion of <i>parentHandle</i> is loaded; or <i>objectPublic</i> and <i>duplicate</i> are of different types
TPM_RC_VALUE	nonempty <i>inSymSeed</i> and its numeric value is greater than the modulus of the key referenced by <i>parentHandle</i> or <i>inSymSeed</i> is larger than the size of the digest produced by the name algorithm of the symmetric key referenced by <i>parentHandle</i>

```

4  TPM_RC
5  TPM2_Import(
6      Import_In      *in,          // IN: input parameter list
7      Import_Out     *out         // OUT: output parameter list
8  )
9  {
10
11      TPM_RC          result = TPM_RC_SUCCESS;
12      OBJECT         *parentObject;
13      TPM2B_DATA      data;        // symmetric key
14      TPMT_SENSITIVE sensitive;
15      TPM2B_NAME      name;
16
17      UINT16          innerKeySize = 0; // encrypt key size for inner
18                                          // wrapper
19
20  // Input Validation
21
22  // FixedTPM and fixedParent must be CLEAR
23  if( in->objectPublic.t.publicArea.objectAttributes.fixedTPM == SET
24      || in->objectPublic.t.publicArea.objectAttributes.fixedParent == SET)
25      return TPM_RC_ATTRIBUTES + RC_Import_objectPublic;
26
27  // Get parent pointer
28  parentObject = ObjectGet(in->parentHandle);
29
30  if(!AreAttributesForParent(parentObject))
31      return TPM_RC_TYPE + RC_Import_parentHandle;
32
33  if(in->symmetricAlg.algorithm != TPM_ALG_NULL)
34  {
35      // Get inner wrap key size
36      innerKeySize = in->symmetricAlg.keyBits.sym;
37      // Input symmetric key must match the size of algorithm.
38      if(in->encryptionKey.t.size != (innerKeySize + 7) / 8)
39          return TPM_RC_SIZE + RC_Import_encryptionKey;
40  }
41  else
42  {
43      // If input symmetric algorithm is NULL, input symmetric key size must
44      // be 0 as well
45      if(in->encryptionKey.t.size != 0)
46          return TPM_RC_SIZE + RC_Import_encryptionKey;
47  }
48
49  // See if there is an outer wrapper
50  if(in->inSymSeed.t.size != 0)
51  {
52      // Decrypt input secret data via asymmetric decryption. TPM_RC_ATTRIBUTES,
53      // TPM_RC_ECC_POINT, TPM_RC_INSUFFICIENT, TPM_RC_KEY, TPM_RC_NO_RESULT,
54      // TPM_RC_SIZE, TPM_RC_VALUE may be returned at this point
55      result = CryptSecretDecrypt(in->parentHandle, NULL, "DUPLICATE",
56                                  &in->inSymSeed, &data);
57      pAssert(result != TPM_RC_BINDING);
58      if(result != TPM_RC_SUCCESS)
59          return TPM_RC_VALUE + RC_Import_inSymSeed;
60  }
61  else
62  {
63      data.t.size = 0;
64  }
65
66  // Compute name of object
67  ObjectComputeName(&(in->objectPublic.t.publicArea), &name);

```

```
68
69 // Retrieve sensitive from private.
70 // TPM_RC_INSUFFICIENT, TPM_RC_INTEGRITY, TPM_RC_SIZE may be returned here.
71 result = DuplicateToSensitive(&in->duplicate, &name, in->parentHandle,
72                             in->objectPublic.t.publicArea.nameAlg,
73                             (TPM2B_SEED *) &data, &in->symmetricAlg,
74                             &in->encryptionKey, &sensitive);
75 if(result != TPM_RC_SUCCESS)
76     return RcSafeAddToResult(result, RC_Import_duplicate);
77
78 // If the parent of this object has fixedTPM SET, then fully validate this
79 // object so that validation can be skipped when it is loaded
80 if(parentObject->publicArea.objectAttributes.fixedTPM == SET)
81 {
82     TPM_HANDLE      objectHandle;
83
84     // Perform self check on input public area. A TPM_RC_SIZE, TPM_RC_SCHEME,
85     // TPM_RC_VALUE, TPM_RC_SYMMETRIC, TPM_RC_TYPE, TPM_RC_HASH,
86     // TPM_RC_ASYMMETRIC, TPM_RC_ATTRIBUTES or TPM_RC_KDF error may be returned
87     // at this point
88     result = PublicAttributesValidation(TRUE, in->parentHandle,
89                                       &in->objectPublic.t.publicArea);
90     if(result != TPM_RC_SUCCESS)
91         return RcSafeAddToResult(result, RC_Import_objectPublic);
92
93     // Create internal object. A TPM_RC_KEY_SIZE, TPM_RC_KEY or
94     // TPM_RC_OBJECT_MEMORY error may be returned at this point
95     result = ObjectLoad(TPM_RH_NULL, &in->objectPublic.t.publicArea,
96                       &sensitive, NULL, in->parentHandle, FALSE,
97                       &objectHandle);
98     if(result != TPM_RC_SUCCESS)
99         return result;
100
101     // Don't need the object, just needed the checks to be performed so
102     // flush the object
103     ObjectFlush(objectHandle);
104 }
105
106 // Command output
107
108 // Prepare output private data from sensitive
109 SensitiveToPrivate(&sensitive, &name, in->parentHandle,
110                  in->objectPublic.t.publicArea.nameAlg,
111                  &out->outPrivate);
112
113 return TPM_RC_SUCCESS;
114 }
```



## 16 Asymmetric Primitives

### 16.1 Introduction

The commands in this clause provide low-level primitives for access to the asymmetric algorithms implemented in the TPM. Many of these commands are only allowed if the asymmetric key is an unrestricted key.

### 16.2 TPM2\_RSA\_Encrypt

#### 16.2.1 General Description

This command performs RSA encryption using the indicated padding scheme according to PKCS#1v2.1 (PKCS#1). If the *scheme* of *keyHandle* is TPM\_ALG\_NULL, then the caller may use *inScheme* to specify the padding scheme. If *scheme* of *keyHandle* is not TPM\_ALG\_NULL, then *inScheme* shall either be TPM\_ALG\_NULL or be the same as *scheme* (TPM\_RC\_SCHEME).

The key referenced by *keyHandle* is required to be an RSA key (TPM\_RC\_KEY) with the *decrypt* attribute SET (TPM\_RC\_ATTRIBUTES).

NOTE Requiring that the *decrypt* attribute be set allows the TPM to ensure that the scheme selection is done with the presumption that the scheme of the key is a decryption scheme selection. It is understood that this command will operate on a key with only the public part loaded so the caller may modify any key in any desired way. So, this constraint only serves to simplify the TPM logic.

The three types of allowed padding are:

- 1) TPM\_ALG\_OAEP – Data is OAEP padded as described in 7.1 of PKCS#1v2.1. The only supported mask generation is MGF1.
- 2) TPM\_ALG\_RSAES – Data is padded as described in 7.2 of PKCS#1v2.1.
- 3) TPM\_ALG\_NULL – Data is not padded by the TPM and the TPM will treat *message* as an unsigned integer and perform a modular exponentiation of *message* using the public exponent of the key referenced by *keyHandle*. This scheme is only used if both the *scheme* in the key referenced by *keyHandle* is TPM\_ALG\_NULL, and the *inScheme* parameter of the command is TPM\_ALG\_NULL. The input value cannot be larger than the public modulus of the key referenced by *keyHandle*.

**Table 41 — Padding Scheme Selection**

<i>keyHandle</i> → <i>scheme</i>	<i>inScheme</i>	padding scheme used
TPM_ALG_NULL	TPM_ALG_NULL	none
	TPM_ALG_RSAES	RSAES
	TPM_ALG_OAEP	OAEP
TPM_ALG_RSAES	TPM_ALG_NULL	RSAES
	TPM_ALG_RSAES	RSAES
	TPM_ALG_OAEP	error (TPM_RC_SCHEME)
TPM_ALG_OAEP	TPM_ALG_NULL	OAEP
	TPM_ALG_RSAES	error (TPM_RC_SCHEME)
	TPM_AGL_OAEP	OAEP

After padding, the data is RSAEP encrypted according to 5.1.1 of PKCS#1v2.1.

NOTE 1 It is required that *decrypt* be SET so that the commands that load a key can validate that the scheme is consistent rather than have that deferred until the key is used.

NOTE 2 If it is desired to use a key that had restricted SET, the caller may CLEAR restricted and load the public part of the key and use that unrestricted version of the key for encryption.

If *inScheme* is used, and the scheme requires a hash algorithm it may not be TPM\_ALG\_NULL.

NOTE 3 Because only the public portion of the key needs to be loaded for this command, the caller can manipulate the attributes of the key in any way desired. As a result, the TPM shall not check the consistency of the attributes. The only property checking is that the key is an RSA key and that the padding scheme is supported.

The *message* parameter is limited in size by the padding scheme according to the following table:

**Table 42 — Message Size Limits Based on Padding**

Scheme	Maximum Message Length ( <i>mLen</i> ) in Octets	Comments
TPM_ALG_OAEP	$mLen \leq k - 2hLen - 2$	
TPM_ALG_RSAES	$mLen \leq k - 11$	
TPM_ALG_NULL	$mLen \leq k$	The numeric value of the message must be less than the numeric value of the public modulus ( <i>n</i> ).
NOTES		
1) <i>k</i> := the number of bytes in the public modulus		
2) <i>hLen</i> := the number of octets in the digest produced by the hash algorithm used in the process		

The *label* parameter is optional. If provided (*label.size* != 0) then the TPM shall return TPM\_RC\_VALUE if the last octet in *label* is not zero. If a zero octet occurs before *label.buffer[label.size-1]*, the TPM shall truncate the label at that point. The terminating octet of zero is included in the *label* used in the padding scheme.

NOTE 4 If the scheme does not use a label, the TPM will still verify that label is properly formatted if label is present.

The function returns padded and encrypted value *outData*.

The *message* parameter in the command may be encrypted using parameter encryption.

NOTE 5 Only the public area of *keyHandle* is required to be loaded. A public key may be loaded with any desired scheme. If the scheme is to be changed, a different public area must be loaded.

16.2.2 Command and Response

Table 43 — TPM2\_RSA\_Encrypt Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_RSA_Encrypt
TPMI_DH_OBJECT	keyHandle	reference to public portion of RSA key to use for encryption Auth Index: None
TPM2B_PUBLIC_KEY_RSA	message	message to be encrypted NOTE 1 The data type was chosen because it limits the overall size of the input to no greater than the size of the largest RSA public key. This may be larger than allowed for <i>keyHandle</i> .
TPMT_RSA_DECRYPT+	inScheme	the padding scheme to use if <i>scheme</i> associated with <i>keyHandle</i> is TPM_ALG_NULL
TPM2B_DATA	label	optional label <i>L</i> to be associated with the message Size of the buffer is zero if no label is present NOTE 2 See description of label above.

Table 44 — TPM2\_RSA\_Encrypt Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PUBLIC_KEY_RSA	outData	encrypted output

## 16.2.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "RSA_Encrypt_fp.h"
3  #ifdef TPM_ALG_RSA

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	<i>decrypt</i> attribute is not SET in key referenced by <i>keyHandle</i>
TPM_RC_KEY	<i>keyHandle</i> does not reference an RSA key
TPM_RC_SCHEME	incorrect input scheme, or the chosen scheme is not a valid RSA decrypt scheme
TPM_RC_VALUE	the numeric value of <i>message</i> is greater than the public modulus of the key referenced by <i>keyHandle</i> , or <i>label</i> is not a null-terminated string

```

4  TPM_RC
5  TPM2_RSA_Encrypt(
6      RSA_Encrypt_In      *in,          // IN: input parameter list
7      RSA_Encrypt_Out     *out         // OUT: output parameter list
8  )
9  {
10     TPM_RC                result;
11     OBJECT                *rsaKey;
12     TPMT_RSA_DECRYPT      *scheme;
13     char                  *label = NULL;
14
15     // Input Validation
16
17     rsaKey = ObjectGet(in->keyHandle);
18
19     // selected key must be an RSA key
20     if(rsaKey->publicArea.type != TPM_ALG_RSA)
21         return TPM_RC_KEY + RC_RSA_Encrypt_keyHandle;
22
23     // selected key must have the decryption attribute
24     if(rsaKey->publicArea.objectAttributes.decrypt != SET)
25         return TPM_RC_ATTRIBUTES + RC_RSA_Encrypt_keyHandle;
26
27     // Is there a label?
28     if(in->label.t.size > 0)
29     {
30         // label is present, so make sure that is it NULL-terminated
31         if(in->label.t.buffer[in->label.t.size - 1] != 0)
32             return TPM_RC_VALUE + RC_RSA_Encrypt_label;
33         label = (char *)in->label.t.buffer;
34     }
35
36     // Command Output
37
38     // Select a scheme for encryption
39     scheme = CryptSelectRSAScheme(in->keyHandle, &in->inScheme);
40     if(scheme == NULL)
41         return TPM_RC_SCHEME + RC_RSA_Encrypt_inScheme;
42
43     // Encryption. TPM_RC_VALUE, or TPM_RC_SCHEME errors may be returned by
44     // CryptEncryptRSA. Note: It can also return TPM_RC_ATTRIBUTES if the key does
45     // not have the decrypt attribute but that was checked above.
46     out->outData.t.size = sizeof(out->outData.t.buffer);
47     result = CryptEncryptRSA(&out->outData.t.size, out->outData.t.buffer, rsaKey,

```

```
48                                     scheme, in->message.t.size, in->message.t.buffer,  
49                                     label);  
50     return result;  
51 }  
52 #endif
```

## 16.3 TPM2\_RSA\_Decrypt

### 16.3.1 General Description

This command performs RSA decryption using the indicated padding scheme according to PKCS#1v2.1 (PKCS#1).

The scheme selection for this command is the same as for TPM2\_RSA\_Encrypt() and is shown in Table 41.

The key referenced by *keyHandle* shall be an RSA key (TPM\_RC\_KEY) with *restricted* CLEAR and *decrypt* SET (TPM\_RC\_ATTRIBUTES).

This command uses the private key of *keyHandle* for this operation and authorization is required.

The TPM will perform a modular exponentiation of ciphertext using the private exponent associated with *keyHandle* (this is described in PKCS#1v2.1, clause 5.1.2). It will then validate the padding according to the selected scheme. If the padding checks fail, TPM\_RC\_VALUE is returned. Otherwise, the data is returned with the padding removed. If no padding is used, the returned value is an unsigned integer value that is the result of the modular exponentiation of *cipherText* using the private exponent of *keyHandle*. The returned value may include leading octets zeros so that it is the same size as the public modulus. For the other padding schemes, the returned value will be smaller than the public modulus but will contain all the data remaining after padding is removed and this may include leading zeros if the original encrypted value contained leading zeros..

If a label is used in the padding process of the scheme, the *label* parameter is required to be present in the decryption process and *label* is required to be the same in both cases. The TPM shall verify that the label is consistent and if not it shall return TPM\_RC\_VALUE. If *label* is present (*label.size* != 0), it shall be a NULL-terminated string or the TPM will return TPM\_RC\_VALUE.

NOTE 1            The size of *label* includes the terminating null.

The *message* parameter in the response may be encrypted using parameter encryption.

If the decryption scheme does not require a hash function, the *hash* parameter of *inScheme* may be set to any valid hash function or TPM\_ALG\_NULL.

If the description scheme does not require a label, the value in *label* is not used but the size of the label field is checked for consistency with the indicated data type (TPM2B\_DATA). That is, the field may not be larger than allowed for a TPM2B\_DATA.

16.3.2 Command and Response

Table 45 — TPM2\_RSA\_Decrypt Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_RSA_Decrypt
TPMI_DH_OBJECT	@keyHandle	RSA key to use for decryption Auth Index: 1 Auth Role: USER
TPM2B_PUBLIC_KEY_RSA	cipherText	cipher text to be decrypted NOTE An encrypted RSA data block is the size of the public modulus.
TPMT_RSA_DECRYPT+	inScheme	the padding scheme to use if <i>scheme</i> associated with <i>keyHandle</i> is TPM_ALG_NULL
TPM2B_DATA	label	label whose association with the message is to be verified

Table 46 — TPM2\_RSA\_Decrypt Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PUBLIC_KEY_RSA	message	decrypted output

## 16.3.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "RSA_Decrypt_fp.h"
3  #ifdef TPM_ALG_RSA

```

Error Returns	Meaning
TPM_RC_KEY	<i>keyHandle</i> does not reference an unrestricted decrypt key
TPM_RC_SCHEME	incorrect input scheme, or the chosen <i>scheme</i> is not a valid RSA decrypt scheme
TPM_RC_SIZE	<i>cipherText</i> is not the size of the modulus of key referenced by <i>keyHandle</i>
TPM_RC_VALUE	<i>label</i> is not a null terminated string or the value of <i>cipherText</i> is greater than the modulus of <i>keyHandle</i>

```

4  TPM_RC
5  TPM2_RSA_Decrypt(
6      RSA_Decrypt_In          *in,          // IN: input parameter list
7      RSA_Decrypt_Out        *out          // OUT: output parameter list
8  )
9  {
10     TPM_RC                    result;
11     OBJECT                    *rsaKey;
12     TPMT_RSA_DECRYPT          *scheme;
13     char                       *label = NULL;
14
15     // Input Validation
16
17     rsaKey = ObjectGet(in->keyHandle);
18
19     // The selected key must be an RSA key
20     if(rsaKey->publicArea.type != TPM_ALG_RSA)
21         return TPM_RC_KEY + RC_RSA_Decrypt_keyHandle;
22
23     // The selected key must be an unrestricted decryption key
24     if( rsaKey->publicArea.objectAttributes.restricted == SET
25        || rsaKey->publicArea.objectAttributes.decrypt == CLEAR)
26         return TPM_RC_ATTRIBUTES + RC_RSA_Decrypt_keyHandle;
27
28     // NOTE: Proper operation of this command requires that the sensitive area
29     // of the key is loaded. This is assured because authorization is required
30     // to use the sensitive area of the key. In order to check the authorization,
31     // the sensitive area has to be loaded, even if authorization is with policy.
32
33     // If label is present, make sure that it is a NULL-terminated string
34     if(in->label.t.size > 0)
35     {
36         // Present, so make sure that it is NULL-terminated
37         if(in->label.t.buffer[in->label.t.size - 1] != 0)
38             return TPM_RC_VALUE + RC_RSA_Decrypt_label;
39         label = (char *)in->label.t.buffer;
40     }
41
42     // Command Output
43
44     // Select a scheme for decrypt.
45     scheme = CryptSelectRSAScheme(in->keyHandle, &in->inScheme);
46     if(scheme == NULL)
47         return TPM_RC_SCHEME + RC_RSA_Decrypt_inScheme;
48

```



```
49     // Decryption.  TPM_RC_VALUE, TPM_RC_SIZE, and TPM_RC_KEY error may be
50     // returned by CryptDecryptRSA.
51     // NOTE: CryptDecryptRSA can also return TPM_RC_ATTRIBUTES or TPM_RC_BINDING
52     // when the key is not a decryption key but that was checked above.
53     out->message.t.size = sizeof(out->message.t.buffer);
54     result = CryptDecryptRSA(&out->message.t.size, out->message.t.buffer, rsaKey,
55                             scheme, in->cipherText.t.size,
56                             in->cipherText.t.buffer,
57                             label);
58
59     return result;
60 }
61 #endif
```

## 16.4 TPM2\_ECDH\_KeyGen

### 16.4.1 General Description

This command uses the TPM to generate an ephemeral key pair  $(d_e, Q_e$  where  $Q_e := [d_e]G$ ). It uses the private ephemeral key and a loaded public key  $(Q_s)$  to compute the shared secret value  $(P := [hd_e]Q_s)$ .

*KeyHandle* shall refer to a loaded ECC key. The sensitive portion of this key need not be loaded.

The curve parameters of the loaded ECC key are used to generate the ephemeral key.

NOTE 1            This function is the equivalent of encrypting data to another object's public key. The *seed* value is used in a KDF to generate a symmetric key and that key is used to encrypt the data. Once the data is encrypted and the symmetric key discarded, only the object with the private portion of the *keyHandle* will be able to decrypt it.

The *zPoint* in the response may be encrypted using parameter encryption.

16.4.2 Command and Response

Table 47 — TPM2\_ECDH\_KeyGen Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ECDH_KeyGen
TPMI_DH_OBJECT	keyHandle	Handle of a loaded ECC key public area. Auth Index: None

Table 48 — TPM2\_ECDH\_KeyGen Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	zPoint	results of $P := h[d_e]Q_s$
TPM2B_ECC_POINT	pubPoint	generated ephemeral public point ( $Q_e$ )

## 16.4.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "ECDH_KeyGen_fp.h"
3  #ifdef TPM_ALG_ECC

```

Error Returns	Meaning
TPM_RC_KEY	<i>keyHandle</i> does not reference a non-restricted decryption ECC key

```

4  TPM_RC
5  TPM2_ECDH_KeyGen(
6      ECDH_KeyGen_In    *in,           // IN: input parameter list
7      ECDH_KeyGen_Out   *out          // OUT: output parameter list
8  )
9  {
10     OBJECT                *eccKey;
11     TPM2B_ECC_PARAMETER   sensitive;
12     TPM_RC                result;
13
14     // Input Validation
15
16     eccKey = ObjectGet(in->keyHandle);
17
18     // Input key must be a non-restricted, decrypt ECC key
19     if( eccKey->publicArea.type != TPM_ALG_ECC
20         || eccKey->publicArea.objectAttributes.restricted == SET
21         || eccKey->publicArea.objectAttributes.decrypt != SET
22     )
23         return TPM_RC_KEY + RC_ECDH_KeyGen_keyHandle;
24
25     // Command Output
26     do
27     {
28         // Create ephemeral ECC key
29         CryptNewEccKey(eccKey->publicArea.parameters.eccDetail.curveID,
30                       &out->pubPoint.t.point, &sensitive);
31
32         out->pubPoint.t.size = TPMS_ECC_POINT_Marshal(&out->pubPoint.t.point,
33                                                       NULL, NULL);
34
35         // Compute Z
36         result = CryptEccPointMultiply(&out->zPoint.t.point,
37                                       eccKey->publicArea.parameters.eccDetail.curveID,
38                                       &sensitive, &eccKey->publicArea.unique.ecc);
39         // The point in the key is not on the curve. Indicate that the key is bad.
40         if(result == TPM_RC_ECC_POINT)
41             return TPM_RC_KEY + RC_ECDH_KeyGen_keyHandle;
42         // The other possible error is TPM_RC_NO_RESULT indicating that the
43         // multiplication resulted in the point at infinity, so get a new
44         // random key and start over (hardly ever happens).
45     }
46     while(result != TPM_RC_SUCCESS);
47
48     // Marshal the values to generate the point.
49     out->zPoint.t.size = TPMS_ECC_POINT_Marshal(&out->zPoint.t.point, NULL, NULL);
50
51     return TPM_RC_SUCCESS;
52 }
53 #endif

```

## 16.5 TPM2\_ECDH\_ZGen

### 16.5.1 General Description

This command uses the TPM to recover the  $Z$  value from a public point ( $Q_B$ ) and a private key ( $d_s$ ). It will perform the multiplication of the provided *inPoint* ( $Q_B$ ) with the private key ( $d_s$ ) and return the coordinates of the resultant point ( $Z = (x_Z, y_Z) := [hd_s]Q_B$ ; where  $h$  is the cofactor of the curve).

*keyHandle* shall refer to a loaded, ECC key (TPM\_RC\_KEY) with the *restricted* attribute CLEAR and the *decrypt* attribute SET (TPM\_RC\_ATTRIBUTES).

The *scheme* of the key referenced by *keyHandle* is required to be either TPM\_ALG\_ECDH or TPM\_ALG\_NULL (TPM\_RC\_SCHEME).

*inPoint* is required to be on the curve of the key referenced by *keyHandle* (TPM\_RC\_ECC\_POINT).

The parameters of the key referenced by *keyHandle* are used to perform the point multiplication.

## 16.5.2 Command and Response

Table 49 — TPM2\_ECDH\_ZGen Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ECDH_ZGen
TPMI_DH_OBJECT	@keyHandle	handle of a loaded ECC key Auth Index: 1 Auth Role: USER
TPM2B_ECC_POINT	inPoint	a public key

Table 50 — TPM2\_ECDH\_ZGen Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	outPoint	X and Y coordinates of the product of the multiplication $Z = (x_Z, y_Z) := [hd_s]Q_B$

## 16.5.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "ECDH_ZGen_fp.h"
3  #ifdef TPM_ALG_ECC

```

Error Returns	Meaning
TPM_RC_KEY	<i>keyHandle</i> does not reference a non-restricted decryption ECC key
TPM_RC_ECC_POINT	invalid argument
TPM_RC_NO_RESULT	multiplying <i>inPoint</i> resulted in a point at infinity

```

4  TPM_RC
5  TPM2_ECDH_ZGen(
6      ECDH_ZGen_In    *in,           // IN: input parameter list
7      ECDH_ZGen_Out   *out          // OUT: output parameter list
8  )
9  {
10     TPM_RC          result;
11     OBJECT          *eccKey;
12
13     // Input Validation
14
15     eccKey = ObjectGet(in->keyHandle);
16
17     // Input key must be a non-restricted, decrypt ECC key
18     if( eccKey->publicArea.type != TPM_ALG_ECC
19         || eccKey->publicArea.objectAttributes.restricted == SET
20         || eccKey->publicArea.objectAttributes.decrypt != SET
21     )
22         return TPM_RC_KEY + RC_ECDH_ZGen_keyHandle;
23
24     // Command Output
25
26     // Compute Z. TPM_RC_ECC_POINT or TPM_RC_NO_RESULT may be returned here.
27     result = CryptEccPointMultiply(&out->outPoint.t.point,
28                                   eccKey->publicArea.parameters.eccDetail.curveID,
29                                   &eccKey->sensitive.sensitive.ecc,
30                                   &in->inPoint.t.point);
31     if(result != TPM_RC_SUCCESS)
32         return RcSafeAddToResult(result, RC_ECDH_ZGen_inPoint);
33
34     out->outPoint.t.size = TPMS_ECC_POINT_Marshal(&out->outPoint.t.point,
35                                                  NULL, NULL);
36
37     return TPM_RC_SUCCESS;
38 }
39 #endif

```

## 16.6 TPM2\_ECC\_Parameters

### 16.6.1 General Description

This command returns the parameters of an ECC curve identified by its TCG-assigned *curveID*.

### 16.6.2 Command and Response

**Table 51 — TPM2\_ECC\_Parameters Command**

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ECC_Parameters
TPMI_ECC_CURVE	curveID	parameter set selector

**Table 52 — TPM2\_ECC\_Parameters Response**

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPMS_ALGORITHM_DETAIL_ECC	parameters	ECC parameters for the selected curve



16.6.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "ECC_Parameters_fp.h"
3  #ifdef TPM_ALG_ECC

```

Error Returns	Meaning
TPM_RC_VALUE	Unsupported ECC curve ID

```

4  TPM_RC
5  TPM2_ECC_Parameters(
6      ECC_Parameters_In    *in,           // IN: input parameter list
7      ECC_Parameters_Out  *out          // OUT: output parameter list
8  )
9  {
10 // Command Output
11
12 // Get ECC curve parameters
13 if(CryptEccGetParameters(in->curveID, &out->parameters))
14     return TPM_RC_SUCCESS;
15 else
16     return TPM_RC_VALUE + RC_ECC_Parameters_curveID;
17 }
18 #endif

```

## 16.1 TPM2\_ZGen\_2Phase

### 16.1.1 General Description

This command supports two-phase key exchange protocols. The command is used in combination with TPM2\_EC\_Ephemeral(). TPM2\_EC\_Ephemeral() generates an ephemeral key and returns the public point of that ephemeral key along with a numeric value that allows the TPM to regenerate the associated private key.

The input parameters for this command are a static public key ( $inQsU$ ), an ephemeral key ( $inQeU$ ) from party B, and the *commitCounter* returned by TPM2\_EC\_Ephemeral(). The TPM uses the counter value to regenerate the ephemeral private key ( $d_{e,v}$ ) and the associated public key ( $Q_{e,v}$ ). *keyA* provides the static ephemeral elements  $d_{s,v}$  and  $Q_{s,v}$ . This provides the two pairs of ephemeral and static keys that are required for the schemes supported by this command.

The TPM will compute  $Z$  or  $Z_s$  and  $Z_e$  according to the selected scheme. If the scheme is not a two-phase key exchange scheme or if the scheme is not supported, the TPM will return TPM\_RC\_SCHEME.

It is an error if  $inQsB$  or  $inQeB$  are not on the curve of *keyA* (TPM\_RC\_ECC\_POINT).

The two-phase key schemes that were assigned an algorithm ID as of the time of the publication of this specification are TPM\_ALG\_ECDH, TPM\_ALG\_ECMQV, and TPM\_ALG\_SM2.

If this command is supported, then support for TPM\_ALG\_ECDH is required. Support for TPM\_ALG\_ECMQV or TPM\_ALG\_SM2 is optional.

NOTE 1 If SM2 is supported and this command is supported, then the implementation is required to support the key exchange protocol of SM2, part 3.

For TPM\_ALG\_ECDH *outZ1* will be  $Z_s$  and *outZ2* will be  $Z_e$  as defined in 6.1.1.2 of SP800-56A.

NOTE 2 A non-restricted decryption key using ECDH may be used in either TPM2\_ECDH\_ZGen() or TPM2\_ZGen\_2Phase as the computation done with the private part of *keyA* is the same in both cases.

For TPM\_ALG\_ECMQV or TPM\_ALG\_SM2 *outZ1* will be  $Z$  and *outZ2* will be an Empty Point.

NOTE 3 An Empty Point has two Empty Buffers as coordinates meaning the minimum *size* value for *outZ2* will be four.

If the input scheme is TPM\_ALG\_ECDH, then *outZ1* will be  $Z_s$  and *outZ2* will be  $Z_e$ . For schemes like MQV (including SM2), *outZ1* will contain the computed value and *outZ2* will be an Empty Point.

NOTE The Z values returned by the TPM are a full point and not just an x-coordinate.

If a computation of either Z produces the point at infinity, then the corresponding Z value will be an Empty Point.

16.1.2 Command and Response

Table 53 — TPM2\_ZGen\_2Phase Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ZGen_2Phase
TPMI_DH_OBJECT	@keyA	handle of an unrestricted decryption key ECC The private key referenced by this handle is used as $d_{S,A}$ Auth Index: 1 Auth Role: USER
TPM2B_ECC_POINT	inQsB	other party's static public key ( $Q_{s,B} = (X_{s,B}, Y_{s,B})$ )
TPM2B_ECC_POINT	inQeB	other party's ephemeral public key ( $Q_{e,B} = (X_{e,B}, Y_{e,B})$ )
TPMI_ECC_KEY_EXCHANGE	inScheme	the key exchange scheme
UINT16	counter	value returned by TPM2_EC_Ephemeral()

Table 54 — TPM2\_ZGen\_2Phase Response

Type	Name	Description
TPM_ST	tag	
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	outZ1	X and Y coordinates of the computed value (scheme dependent)
TPM2B_ECC_POINT	outZ2	X and Y coordinates of the second computed value (scheme dependent)

## 16.1.3 Detailed Actions

```

1 #include "InternalRoutines.h"
2 #include "ZGen_2Phase_fp.h"
3 #if defined TPM_ALG_ECC && (CC_ZGen_2Phase == YES)

```

This command uses the TPM to recover one or two Z values in a two phase key exchange protocol

Error Returns	Meaning
TPM_RC_ATTRIBUTES	key referenced by <i>keyA</i> is restricted or not a decrypt key
TPM_RC_ECC_POINT	<i>inQsB</i> or <i>inQeB</i> is not on the curve of the key reference by <i>keyA</i>
TPM_RC_KEY	key referenced by <i>keyA</i> is not an ECC key
TPM_RC_SCHEME	the scheme of the key referenced by <i>keyA</i> is not TPM_ALG_NULL, TPM_ALG_ECDH, TPM_ALG_ECMQV or TPM_ALG_SM2

```

4 TPM_RC
5 TPM2_ZGen_2Phase(
6     ZGen_2Phase_In    *in,           // IN: input parameter list
7     ZGen_2Phase_Out  *out           // OUT: output parameter list
8 )
9 {
10     TPM_RC            result;
11     OBJECT            *eccKey;
12     TPM2B_ECC_PARAMETER r;
13     TPM_ALG_ID        scheme;
14
15     // Input Validation
16
17     eccKey = ObjectGet(in->keyA);
18
19     // keyA must be an ECC key
20     if(eccKey->publicArea.type != TPM_ALG_ECC)
21         return TPM_RC_KEY + RC_ZGen_2Phase_keyA;
22
23     // keyA must not be restricted and must be a decrypt key
24     if( eccKey->publicArea.objectAttributes.restricted == SET
25         || eccKey->publicArea.objectAttributes.decrypt != SET
26         )
27         return TPM_RC_ATTRIBUTES + RC_ZGen_2Phase_keyA;
28
29     // if the scheme of keyA is TPM_ALG_NULL, then use the input scheme; otherwise
30     // the input scheme must be the same as the scheme of keyA
31     scheme = eccKey->publicArea.parameters.asymDetail.scheme.scheme;
32     if(scheme != TPM_ALG_NULL)
33     {
34         if(scheme != in->inScheme)
35             return TPM_RC_SCHEME + RC_ZGen_2Phase_inScheme;
36     }
37     else
38         scheme = in->inScheme;
39     if(scheme == TPM_ALG_NULL)
40         return TPM_RC_SCHEME + RC_ZGen_2Phase_inScheme;
41
42     // Input points must be on the curve of keyA
43     if(!CryptEccIsPointOnCurve(eccKey->publicArea.parameters.eccDetail.curveID,
44                                &in->inQsB.t.point))
45         return TPM_RC_ECC_POINT + RC_ZGen_2Phase_inQsB;
46
47     if(!CryptEccIsPointOnCurve(eccKey->publicArea.parameters.eccDetail.curveID,

```

```
48         &in->inQeB.t.point)
49     return TPM_RC_ECC_POINT + RC_ZGen_2Phase_inQeB;
50
51     if(!CryptGenerateR(&r, &in->counter,
52         eccKey->publicArea.parameters.eccDetail.curveID,
53         NULL))
54         return TPM_RC_VALUE + RC_ZGen_2Phase_counter;
55
56 // Command Output
57
58     result = CryptEcc2PhaseKeyExchange(&out->outZ1.t.point,
59         &out->outZ2.t.point,
60         eccKey->publicArea.parameters.eccDetail.curveID,
61         scheme,
62         &eccKey->sensitive.sensitive.ecc,
63         &r,
64         &in->inQsB.t.point,
65         &in->inQeB.t.point);
66     if(result != TPM_RC_SUCCESS)
67         return result;
68
69     CryptEndCommit(in->counter);
70
71     return TPM_RC_SUCCESS;
72 }
73 #endif
```

## 17 Symmetric Primitives

### 17.1 Introduction

The commands in this clause provide low-level primitives for access to the symmetric algorithms implemented in the TPM that operate on blocks of data. These include symmetric encryption and decryption as well as hash and HMAC. All of the commands in this group are stateless. That is, they have no persistent state that is retained in the TPM when the command is complete.

For hashing, HMAC, and Events that require large blocks of data with retained state, the sequence commands are provided (see clause 1).

Some of the symmetric encryption/decryption modes use an IV. When an IV is used, it may be an initiation value or a chained value from a previous stage. The chaining for each mode is:

Table 55 — Symmetric Chaining Process

Mode	Chaining process
TPM_ALG_CTR	<p>The TPM will increment the low-order 32 bits of the IV provided by the caller. The last encrypted value will be returned to the caller as <i>IvOut</i>. This can be the input value to the next encrypted buffer.</p> <p><i>IvIn</i> is required to be the size of a block encrypted by the selected algorithm and key combination. If the size of <i>ivIn</i> is not correct, the TPM shall return TPM_RC_SIZE.</p> <p>EXAMPLE 1 AES requires that <i>ivIn</i> be 128 bits (16 octets).</p> <p><i>IvOut</i> will be the size of a cipher block and not the size of the last encrypted block.</p> <p>NOTE <i>IvOut</i> will be the value of the counter after the last block is encrypted.</p> <p>EXAMPLE 2 If <i>ivIn</i> were 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00<sub>16</sub> and four data blocks were encrypted, <i>IvOut</i> will have a value of 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04<sub>16</sub>.</p> <p>All the bits of the IV are incremented as if it were an unsigned integer.</p>
TPM_ALG_OFB	<p>In Output Feedback (OFB), the output of the pseudo-random function (the block encryption algorithm) is XORed with a plaintext block to produce a ciphertext block. <i>IvOut</i> will be the value that was XORed with the last plaintext block. That value can be used as the <i>ivIn</i> for a next buffer.</p> <p><i>IvIn</i> is required to be the size of a block encrypted by the selected algorithm and key combination. If the size of <i>ivIn</i> is not correct, the TPM shall return TPM_RC_SIZE.</p> <p><i>IvOut</i> will be the size of a cipher block and not the size of the last encrypted block.</p>
TPM_ALG_CBC	<p>For Cipher Block Chaining (CBC), a block of ciphertext is XORed with the next plaintext block and that block is encrypted. The encrypted block is then input to the encryption of the next block. The last ciphertext block then is used as an IV for the next buffer.</p> <p>Even though the last ciphertext block is evident in the encrypted data, it is also returned in <i>IvOut</i>.</p> <p><i>IvIn</i> is required to be the size of a block encrypted by the selected algorithm and key combination. If the size of <i>ivIn</i> is not correct, the TPM shall return TPM_RC_SIZE.</p> <p><i>InData</i> is required to be an even multiple of the block encrypted by the selected algorithm and key combination. If the size of <i>inData</i> is not correct, the TPM shall return TPM_RC_SIZE.</p>
TPM_ALG_CFB	<p>Similar to CBC in that the last ciphertext block is an input to the encryption of the next block. <i>IvOut</i> will be the value that was XORed with the last plaintext block. That value can be used as the <i>ivIn</i> for a next buffer.</p> <p><i>IvIn</i> is required to be the size of a block encrypted by the selected algorithm and key combination. If the size of <i>ivIn</i> is not correct, the TPM shall return TPM_RC_SIZE.</p> <p><i>IvOut</i> will be the size of a cipher block and not the size of the last encrypted block.</p>
TPM_ALG_ECB	<p>Electronic Codebook (ECB) has no chaining. Each block of plaintext is encrypted using the key. ECB does not support chaining and <i>ivIn</i> shall be the Empty Buffer. <i>IvOut</i> will be the Empty Buffer.</p> <p><i>InData</i> is required to be an even multiple of the block encrypted by the selected algorithm and key combination. If the size of <i>inData</i> is not correct, the TPM shall return TPM_RC_SIZE.</p>

## 17.2 TPM2\_EncryptDecrypt

### 17.2.1 General Description

This command performs symmetric encryption or decryption encryption.

*Keyhandle* shall reference a symmetric cipher object (TPM\_RC\_KEY).

For a restricted key, *mode* shall be either the same as the mode of the key, or TPM\_ALG\_NULL (TPM\_RC\_VALUE). For an unrestricted key, *mode* may be the same or different from the mode of the key but both shall not be TPM\_ALG\_NULL (TPM\_RC\_VALUE).

If the TPM allows this command to be canceled before completion, then the TPM may produce incremental results and return TPM\_RC\_SUCCESS rather than TPM\_RC\_CANCEL. In such case, *outData* may be less than *inData*.



17.2.2 Command and Response

**Table 56 — TPM2\_EncryptDecrypt Command**

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_EncryptDecrypt
TPMI_DH_OBJECT	@keyHandle	the symmetric key used for the operation Auth Index: 1 Auth Role: USER
TPMI_YES_NO	decrypt	if YES, then the operation is decryption; if NO, the operation is encryption
TPMI_ALG_SYM_MODE+	mode	symmetric mode For a restricted key, this field shall match the default mode of the key or be TPM_ALG_NULL.
TPM2B_IV	ivIn	an initial value as required by the algorithm
TPM2B_MAX_BUFFER	inData	the data to be encrypted/decrypted

**Table 57 — TPM2\_EncryptDecrypt Response**

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_MAX_BUFFER	outData	encrypted output
TPM2B_IV	ivOut	chaining value to use for IV in next round

## 17.2.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "EncryptDecrypt_fp.h"

```

Error Returns	Meaning
TPM_RC_KEY	is not a symmetric decryption key with both public and private portions loaded
TPM_RC_SIZE	<i>ivIn</i> size is incompatible with the block cipher mode; or <i>inData</i> size is not an even multiple of the block size for CBC or ECB mode
TPM_RC_VALUE	<i>keyHandle</i> is restricted and the argument <i>mode</i> does not match the key's mode

```

3  TPM_RC
4  TPM2_EncryptDecrypt(
5      EncryptDecrypt_In      *in,          // IN: input parameter list
6      EncryptDecrypt_Out     *out,        // OUT: output parameter list
7  )
8  {
9      OBJECT                  *symKey;
10     UINT16                   keySize;
11     UINT16                   blockSize;
12     BYTE                     *key;
13     TPM_ALG_ID               alg;
14
15     // Input Validation
16     symKey = ObjectGet(in->keyHandle);
17
18     // The input key should be a symmetric decrypt key.
19     if( symKey->publicArea.type != TPM_ALG_SYMCIPHER
20         || symKey->attributes.publicOnly == SET)
21         return TPM_RC_KEY + RC_EncryptDecrypt_keyHandle;
22
23     // If the input mode is TPM_ALG_NULL, use the key's mode
24     if( in->mode == TPM_ALG_NULL)
25         in->mode = symKey->publicArea.parameters.symDetail.mode.sym;
26
27     // If the key is restricted, the input sym mode should match the key's sym
28     // mode
29     if( symKey->publicArea.objectAttributes.restricted == SET
30         && symKey->publicArea.parameters.symDetail.mode.sym != in->mode)
31         return TPM_RC_VALUE + RC_EncryptDecrypt_mode;
32
33     // If the mode is null, then we have a problem.
34     // Note: Construction of a TPMT_SYM_DEF does not allow the 'mode' to be
35     // TPM_ALG_NULL so setting in->mode to the mode of the key should have
36     // produced a valid mode. However, this is suspenders.
37     if(in->mode == TPM_ALG_NULL)
38         return TPM_RC_VALUE + RC_EncryptDecrypt_mode;
39
40     // The input iv for ECB mode should be null. All the other modes should
41     // have an iv size same as encryption block size
42
43     keySize = symKey->publicArea.parameters.symDetail.keyBits.sym;
44     alg = symKey->publicArea.parameters.symDetail.algorithm;
45     blockSize = CryptGetSymmetricBlockSize(alg, keySize);
46     if( (in->mode == TPM_ALG_ECB && in->IvIn.t.size != 0)
47         || (in->mode != TPM_ALG_ECB && in->IvIn.t.size != blockSize))
48         return TPM_RC_SIZE + RC_EncryptDecrypt_IvIn;
49
50     // The input data size of CBC mode or ECB mode must be an even multiple of

```

```
51     // the symmetric algorithm's block size
52     if( (in->mode == TPM_ALG_CBC || in->mode == TPM_ALG_ECB)
53         && (in->inData.t.size % blockSize) != 0)
54         return TPM_RC_SIZE + RC_EncryptDecrypt_inData;
55
56 // Command Output
57
58     key = symKey->sensitive.sensitive.sym.t.buffer;
59     // For symmetric encryption, the cipher data size is the same as plain data
60     // size.
61     out->outData.t.size = in->inData.t.size;
62     if(in->decrypt == YES)
63     {
64         // Decrypt data to output
65         CryptSymmetricDecrypt(out->outData.t.buffer, alg, keySize, in->mode, key,
66                               &(in->IvIn), in->inData.t.size, in->inData.t.buffer);
67     }
68     else
69     {
70         // Encrypt data to output
71         CryptSymmetricEncrypt(out->outData.t.buffer, alg, keySize, in->mode, key,
72                               &(in->IvIn), in->inData.t.size, in->inData.t.buffer);
73     }
74     // Copy IV
75     out->IvOut = in->IvIn;
76
77     return TPM_RC_SUCCESS;
78 }
```

## 17.3 TPM2\_Hash

### 17.3.1 General Description

This command performs a hash operation on a data buffer and returns the results.

**NOTE** If the data buffer to be hashed is larger than will fit into the TPM's input buffer, then the sequence hash commands will need to be used.

If the results of the hash will be used in a signing operation that uses a restricted signing key, then the ticket returned by this command can indicate that the hash is safe to sign.

If the digest is not safe to sign, then the TPM will return a TPMT\_TK\_HASHCHECK with the hierarchy set to TPM\_RH\_NULL and *digest* set to the Empty Buffer.

If *hierarchy* is TPM\_RH\_NULL, then *digest* in the ticket will be the Empty Buffer.

## 17.3.2 Command and Response

Table 58 — TPM2\_Hash Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	Shall have at least one session
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Hash
TPM2B_MAX_BUFFER	data	data to be hashed
TPMI_ALG_HASH	hashAlg	algorithm for the hash being computed – shall not be TPM_ALG_NULL
TPMI_RH_HIERARCHY+	hierarchy	hierarchy to use for the ticket (TPM_RH_NULL allowed)

Table 59 — TPM2\_Hash Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DIGEST	outHash	results
TPMT_TK_HASHCHECK	validation	ticket indicating that the sequence of octets used to compute <i>outDigest</i> did not start with TPM_GENERATED_VALUE will be a NULL ticket if the digest may not be signed with a restricted key

## 17.3.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Hash_fp.h"
3  TPM_RC
4  TPM2_Hash(
5      Hash_In      *in,           // IN: input parameter list
6      Hash_Out     *out          // OUT: output parameter list
7  )
8  {
9      HASH_STATE     hashState;
10
11  // Command Output
12
13  // Output hash
14  // Start hash stack
15  out->outHash.t.size = CryptStartHash(in->hashAlg, &hashState);
16  // Adding hash data
17  CryptUpdateDigest2B(&hashState, &in->data.b);
18  // Complete hash
19  CryptCompleteHash2B(&hashState, &out->outHash.b);
20
21  // Output ticket
22  out->validation.tag = TPM_ST_HASHCHECK;
23  out->validation.hierarchy = in->hierarchy;
24
25  if(in->hierarchy == TPM_RH_NULL)
26  {
27      // Ticket is not required
28      out->validation.hierarchy = TPM_RH_NULL;
29      out->validation.digest.t.size = 0;
30  }
31  else if( in->data.t.size >= sizeof(TPM_GENERATED)
32          && !TicketIsSafe(&in->data.b))
33  {
34      // Ticket is not safe
35      out->validation.hierarchy = TPM_RH_NULL;
36      out->validation.digest.t.size = 0;
37  }
38  else
39  {
40      // Compute ticket
41      TicketComputeHashCheck(in->hierarchy, &out->outHash, &out->validation);
42  }
43
44  return TPM_RC_SUCCESS;
45  }

```

## 17.4 TPM2\_HMAC

### 17.4.1 General Description

This command performs an HMAC on the supplied data using the indicated hash algorithm.

The caller shall provide proper authorization for use of *handle*.

If the sign attribute is not SET in the key referenced by *handle* then the TPM shall return TPM\_RC\_ATTRIBUTES. If the key type is not TPM\_ALG\_KEYEDHASH then the TPM shall return TPM\_RC\_TYPE.

If *handle* references a restricted key, then the hash algorithm specified in the key's *scheme* is used as the hash algorithm for the HMAC and the TPM shall return TPM\_RC\_VALUE if *hashAlg* is not TPM\_ALG\_NULL or the same algorithm as selected in the key's scheme.

NOTE 1 A restricted key may only have one of sign or decrypt SET and the default scheme may not be TPM\_ALG\_NULL. These restrictions are enforced by TPM2\_Create() and TPM2\_CreatePrimary(),

If the key referenced by *handle* is not restricted, then the TPM will use *hashAlg* for the HMAC. However, if *hashAlg* is TPM\_ALG\_NULL the TPM will use the default scheme of the key.

If both *hashAlg* and the key default are TPM\_ALG\_NULL, the TPM shall return TPM\_RC\_VALUE.

NOTE A key may only have both sign and decrypt SET if the key is unrestricted. When both sign and decrypt are set, there is no default scheme for the key and the hash algorithm must be specified.

## 17.4.2 Command and Response

Table 60 — TPM2\_HMAC Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_HMAC
TPMI_DH_OBJECT	@handle	handle for the symmetric signing key providing the HMAC key Auth Index: 1 Auth Role: USER
TPM2B_MAX_BUFFER	buffer	HMAC data
TPMI_ALG_HASH+	hashAlg	algorithm to use for HMAC

Table 61 — TPM2\_HMAC Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DIGEST	outHMAC	the returned HMAC in a sized buffer



## 17.4.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "HMAC_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	key referenced by <i>handle</i> is not a signing key
TPM_RC_TYPE	key referenced by <i>handle</i> is not an HMAC key
TPM_RC_VALUE	<i>hashAlg</i> specified when the key is restricted is neither TPM_ALG_NULL not equal to that of the key scheme; or both <i>hashAlg</i> and the key scheme's algorithm are TPM_ALG_NULL

```

3  TPM_RC
4  TPM2_HMAC(
5      HMAC_In           *in,           // IN: input parameter list
6      HMAC_Out          *out,          // OUT: output parameter list
7  )
8  {
9      HMAC_STATE         hmacState;
10     OBJECT              *hmacObject;
11     TPMT_ALG_HASH      hashAlg;
12     TPMT_PUBLIC        *publicArea;
13
14     // Input Validation
15
16     // Get HMAC key object and public area pointers
17     hmacObject = ObjectGet(in->handle);
18     publicArea = &hmacObject->publicArea;
19
20     // Make sure that the key is an HMAC signing key
21     if(publicArea->type != TPM_ALG_KEYEDHASH)
22         return TPM_RC_TYPE + RC_HMAC_handle;
23     if(publicArea->objectAttributes.sign != SET)
24         return TPM_RC_ATTRIBUTES + RC_HMAC_handle;
25
26
27     // Assume that the key default scheme is used
28     hashAlg = publicArea->parameters.keyedHashDetail.scheme.details.hmac.hashAlg;
29
30     // if the key is restricted, then need to use the scheme of the key and the
31     // input algorithm must be TPM_ALG_NULL or the same as the key scheme
32     if(publicArea->objectAttributes.restricted == SET)
33     {
34         if(in->hashAlg != TPM_ALG_NULL && in->hashAlg != hashAlg)
35             hashAlg = TPM_ALG_NULL;
36     }
37     else
38     {
39         // for a non-restricted key, use hashAlg if it is provided;
40         if(in->hashAlg != TPM_ALG_NULL)
41             hashAlg = in->hashAlg;
42     }
43     // if the hashAlg is TPM_ALG_NULL, then the input hashAlg is not compatible
44     // with the key scheme or type
45     if(hashAlg == TPM_ALG_NULL)
46         return TPM_RC_VALUE + RC_HMAC_hashAlg;
47
48     // Command Output
49

```

```
50     // Start HMAC stack
51     out->outHMAC.t.size = CryptStartHMAC2B(hashAlg,
52                                           &hmacObject->sensitive.sensitive.bits.b,
53                                           &hmacState);
54     // Adding HMAC data
55     CryptUpdateDigest2B(&hmacState, &in->buffer.b);
56
57     // Complete HMAC
58     CryptCompleteHMAC2B(&hmacState, &out->outHMAC.b);
59
60     return TPM_RC_SUCCESS;
61 }
```

## 18 Random Number Generator

### 18.1 TPM2\_GetRandom

#### 18.1.1 General Description

This command returns the next *bytesRequested* octets from the random number generator (RNG).

NOTE 1 It is recommended that a TPM implement the RNG in a manner that would allow it to return RNG octets such that the frequency of *bytesRequested* being more than the number of octets available is an infrequent occurrence.

If *bytesRequested* is more than will fit into a TPM2B\_DIGEST on the TPM, no error is returned but the TPM will only return as much data as will fit into a TPM2B\_DIGEST buffer for the TPM.

NOTE 2 TPM2B\_DIGEST is large enough to hold the largest digest that may be produced by the TPM. Because that digest size changes according to the implemented hashes, the maximum amount of data returned by this command is TPM implementation-dependent.

## 18.1.2 Command and Response

Table 62 — TPM2\_GetRandom Command

Type	Name	Description
TPML_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_GetRandom
UINT16	bytesRequested	number of octets to return

Table 63 — TPM2\_GetRandom Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DIGEST	randomBytes	the random octets

### 18.1.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "GetRandom_fp.h"
3  TPM_RC
4  TPM2_GetRandom(
5      GetRandom_In      *in,          // IN: input parameter list
6      GetRandom_Out     *out         // OUT: output parameter list
7  )
8  {
9  // Command Output
10
11 // if the requested bytes exceed the output buffer size, generates the
12 // maximum bytes that the output buffer allows
13 if(in->bytesRequested > sizeof(TPMU_HA))
14     out->randomBytes.t.size = sizeof(TPMU_HA);
15 else
16     out->randomBytes.t.size = in->bytesRequested;
17
18 CryptGenerateRandom(out->randomBytes.t.size, out->randomBytes.t.buffer);
19
20 return TPM_RC_SUCCESS;
21 }
```

## 18.2 TPM2\_StirRandom

### 18.2.1 General Description

This command is used to add "additional information" to the RNG state.

NOTE           The "additional information" is as defined in SP800-90A.

The *inData* parameter may not be larger than 128 octets.

## 18.2.2 Command and Response

Table 64 — TPM2\_StirRandom Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_StirRandom {NV}
TPM2B_SENSITIVE_DATA	inData	additional information

Table 65 — TPM2\_StirRandom Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

### 18.2.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "StirRandom_fp.h"
3  TPM_RC
4  TPM2_StirRandom(
5      StirRandom_In    *in                // IN: input parameter list
6  )
7  {
8  // Internal Data Update
9      CryptStirRandom(in->inData.t.size, in->inData.t.buffer);
10
11     return TPM_RC_SUCCESS;
12 }
```



## 19 Hash/HMAC/Event Sequences

### 19.1 Introduction

All of the commands in this group are to support sequences for which an intermediate state must be maintained. For a description of sequences, see “Hash, HMAC, and Event Sequences” in Part 1.

### 19.2 TPM2\_HMAC\_Start

#### 19.2.1 General Description

This command starts an HMAC sequence. The TPM will create and initialize an HMAC sequence structure, assign a handle to the sequence, and set the *authValue* of the sequence object to the value in *auth*.

NOTE 1 The structure of a sequence object is vendor-dependent.

The caller shall provide proper authorization for use of *handle*.

If the *sign* attribute is not SET in the key referenced by *handle* then the TPM shall return TPM\_RC\_ATTRIBUTES. If the key type is not TPM\_ALG\_KEYEDHASH then the TPM shall return TPM\_RC\_TYPE.

If *handle* references a restricted key, then the hash algorithm specified in the key's *scheme* is used as the hash algorithm for the HMAC and the TPM shall return TPM\_RC\_VALUE if *hashAlg* is not TPM\_ALG\_NULL or the same algorithm in the key's *scheme*.

If the key referenced by *handle* is not restricted, then the TPM will use *hashAlg* for the HMAC; unless *hashAlg* is TPM\_ALG\_NULL in which case it will use the default scheme of the key.

**Table 66 — Hash Selection Matrix**

<i>handle</i> → <i>restricted</i> (key's restricted attribute)	<i>handle</i> → <i>scheme</i> (hash algorithm from key's <i>scheme</i> )	<i>hashAlg</i>	hash used
CLEAR (unrestricted)	TPM_ALG_NULL <sup>(1)</sup>	TPM_ALG_NULL	error <sup>(2)</sup> (TPM_RC_SCHEME)
CLEAR	don't care	valid hash	<i>hashAlg</i>
CLEAR	valid hash	TPM_ALG_NULL	<i>handle</i> → <i>scheme</i>
SET (restricted)	valid hash <sup>(3)</sup>	TPM_ALG_NULL	<i>handle</i> → <i>scheme</i>
SET	valid hash <sup>(3)</sup>	same as <i>handle</i> → <i>scheme</i>	<i>handle</i> → <i>scheme</i>
SET	valid hash <sup>(3)</sup>	not same as <i>handle</i> → <i>scheme</i>	error <sup>(4)</sup> (TPM_RC_SCHEME)

NOTES:

- 1) The scheme for the handle may only be TPM\_ALG\_NULL if both *sign* and *decrypt* are SET.
- 2) A hash algorithm is required for the HMAC.
- 3) A restricted key is required to have a scheme with a valid hash algorithm. A restricted key may not have both *sign* and *decrypt* SET.
- 4) The scheme for a restricted key cannot be overridden.

## 19.2.2 Command and Response

Table 67 — TPM2\_HMAC\_Start Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_HMAC_Start
TPMI_DH_OBJECT+	@handle	handle of an HMAC key Auth Index: 1 Auth Role: USER
TPM2B_AUTH	auth	authorization value for subsequent use of the sequence
TPMI_ALG_HASH+	hashAlg	the hash algorithm to use for the HMAC

Table 68 — TPM2\_HMAC\_Start Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPMI_DH_OBJECT	sequenceHandle	a handle to reference the sequence

## 19.2.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "HMAC_Start_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	key referenced by <i>handle</i> is not a signing key
TPM_RC_OBJECT_MEMORY	no space to create an internal object
TPM_RC_TYPE	key referenced by <i>handle</i> is not an HMAC key
TPM_RC_VALUE	<i>hashAlg</i> specified when the key is restricted is neither TPM_ALG_NULL not equal to that of the key scheme; or both <i>hashAlg</i> and the key scheme's algorithm are TPM_ALG_NULL

```

3  TPM_RC
4  TPM2_HMAC_Start(
5      HMAC_Start_In      *in,          // IN: input parameter list
6      HMAC_Start_Out     *out         // OUT: output parameter list
7  )
8  {
9      OBJECT              *hmacObject;
10     TPMT_PUBLIC          *publicArea;
11     TPM_ALG_ID           hashAlg;
12
13     // Input Validation
14
15     // Get HMAC key object and public area pointers
16     hmacObject = ObjectGet(in->handle);
17     publicArea = &hmacObject->publicArea;
18
19     // Make sure that the key is an HMAC signing key
20     if(publicArea->type != TPM_ALG_KEYEDHASH)
21         return TPM_RC_TYPE + RC_HMAC_Start_handle;
22     if(publicArea->objectAttributes.sign != SET)
23         return TPM_RC_ATTRIBUTES + RC_HMAC_Start_handle;
24
25     // Assume that the key default scheme is used
26     hashAlg = publicArea->parameters.keyedHashDetail.scheme.details.hmac.hashAlg;
27
28     // if the key is restricted, then need to use the scheme of the key and the
29     // input algorithm must be TPM_ALG_NULL or the same as the key scheme
30     if(publicArea->objectAttributes.restricted == SET)
31     {
32         if(in->hashAlg != TPM_ALG_NULL && in->hashAlg != hashAlg)
33             hashAlg = TPM_ALG_NULL;
34     }
35     else
36     {
37         // for a non-restricted key, use hashAlg if it is provided;
38         if(in->hashAlg != TPM_ALG_NULL)
39             hashAlg = in->hashAlg;
40     }
41     // if the algorithm selection ended up with TPM_ALG_NULL, then either the
42     // schemes are not compatible or no hash was provided and both conditions
43     // are errors.
44     if(hashAlg == TPM_ALG_NULL)
45         return TPM_RC_VALUE + RC_HMAC_Start_hashAlg;
46
47     // Internal Data Update

```

```
48
49 // Create a HMAC sequence object. A TPM_RC_OBJECT_MEMORY error may be
50 // returned at this point
51 return ObjectCreateHMACSequence(hashAlg,
52                                 in->handle,
53                                 &in->auth,
54                                 &out->sequenceHandle);
55 }
```

### 19.3 TPM2\_HashSequenceStart

#### 19.3.1 General Description

This command starts a hash or an Event sequence. If *hashAlg* is an implemented hash, then a hash sequence is started. If *hashAlg* is TPM\_ALG\_NULL, then an Event sequence is started. If *hashAlg* is neither an implemented algorithm nor TPM\_ALG\_NULL, then the TPM shall return TPM\_RC\_HASH.

Depending on *hashAlg*, the TPM will create and initialize a hash sequence structure or an Event sequence structure. Additionally, it will assign a handle to the sequence and set the *authValue* of the sequence to the value in *auth*. A sequence structure for an Event (*hashAlg* = TPM\_ALG\_NULL) contains a hash context for each of the PCR banks implemented on the TPM.

## 19.3.2 Command and Response

Table 69 — TPM2\_HashSequenceStart Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_HashSequenceStart
TPM2B_AUTH	auth	authorization value for subsequent use of the sequence
TPMI_ALG_HASH+	hashAlg	the hash algorithm to use for the hash sequence An Event sequence starts if this is TPM_ALG_NULL.

Table 70 — TPM2\_HashSequenceStart Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPMI_DH_OBJECT	sequenceHandle	a handle to reference the sequence

## 19.3.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "HashSequenceStart_fp.h"

```

Error Returns	Meaning
TPM_RC_OBJECT_MEMORY	no space to create an internal object

```

3  TPM_RC
4  TPM2_HashSequenceStart(
5      HashSequenceStart_In      *in,          // IN: input parameter list
6      HashSequenceStart_Out     *out,          // OUT: output parameter list
7  )
8  {
9      // Internal Data Update
10
11     if(in->hashAlg == TPM_ALG_NULL)
12         // Start a event sequence. A TPM_RC_OBJECT_MEMORY error may be
13         // returned at this point
14         return ObjectCreateEventSequence(&in->auth, &out->sequenceHandle);
15
16     // Start a hash sequence. A TPM_RC_OBJECT_MEMORY error may be
17     // returned at this point
18     return ObjectCreateHashSequence(in->hashAlg, &in->auth, &out->sequenceHandle);
19 }

```

## 19.4 TPM2\_SequenceUpdate

### 19.4.1 General Description

This command is used to add data to a hash or HMAC sequence. The amount of data in buffer may be any size up to the limits of the TPM.

NOTE In all TPM, a *buffer* size of 1,024 octets is allowed.

Proper authorization for the sequence object associated with *sequenceHandle* is required. If an authorization or audit of this command requires computation of a *cpHash* and an *rpHash*, the Name associated with *sequenceHandle* will be the Empty Buffer.

If the command does not return TPM\_RC\_SUCCESS, the state of the sequence is unmodified.

If the sequence is intended to produce a digest that will be signed by a restricted signing key, then the first block of data shall contain sizeof(TPM\_GENERATED) octets and the first octets shall not be TPM\_GENERATED\_VALUE.

NOTE This requirement allows the TPM to validate that the first block is safe to sign without having to accumulate octets over multiple calls.



19.4.2 Command and Response

**Table 71 — TPM2\_SequenceUpdate Command**

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_SequenceUpdate
TPMI_DH_OBJECT	@sequenceHandle	handle for the sequence object Auth Index: 1 Auth Role: USER
TPM2B_MAX_BUFFER	buffer	data to be added to hash

**Table 72 — TPM2\_SequenceUpdate Response**

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 19.4.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "SequenceUpdate_fp.h"

```

Error Returns	Meaning
TPM_RC_MODE	<i>sequenceHandle</i> does not reference a hash or HMAC sequence object

```

3  TPM_RC
4  TPM2_SequenceUpdate(
5      SequenceUpdate_In      *in          // IN: input parameter list
6  )
7  {
8      OBJECT                  *object;
9
10     // Input Validation
11
12     // Get sequence object pointer
13     object = ObjectGet(in->sequenceHandle);
14
15     // Check that referenced object is a sequence object.
16     if(!ObjectIsSequence(object))
17         return TPM_RC_MODE + RC_SequenceUpdate_sequenceHandle;
18
19     // Internal Data Update
20
21     if(object->attributes.eventSeq == SET)
22     {
23         // Update event sequence object
24         UINT32      i;
25         HASH_OBJECT *hashObject = (HASH_OBJECT *)object;
26         for(i = 0; i < HASH_COUNT; i++)
27         {
28             // Update sequence object
29             CryptUpdateDigest2B(&hashObject->state.hashState[i], &in->buffer.b);
30         }
31     }
32     else
33     {
34         HASH_OBJECT *hashObject = (HASH_OBJECT *)object;
35
36         // Update hash/HMAC sequence object
37         if(hashObject->attributes.hashSeq == SET)
38         {
39             // Is this the first block of the sequence
40             if(hashObject->attributes.firstBlock == CLEAR)
41             {
42                 // If so, indicate that first block was received
43                 hashObject->attributes.firstBlock = SET;
44
45                 // Check the first block to see if the first block can contain
46                 // the TPM_GENERATED_VALUE.  If it does, it is not safe for
47                 // a ticket.
48                 if(TicketIsSafe(&in->buffer.b))
49                     hashObject->attributes.ticketSafe = SET;
50             }
51             // Update sequence object hash/HMAC stack
52             CryptUpdateDigest2B(&hashObject->state.hashState[0], &in->buffer.b);
53
54         }

```

```
55     else if(object->attributes.hmacSeq == SET)
56     {
57         HASH_OBJECT      *hashObject = (HASH_OBJECT *)object;
58
59         // Update sequence object hash/HMAC stack
60         CryptUpdateDigest2B(&hashObject->state.hmacState, &in->buffer.b);
61     }
62 }
63
64 return TPM_RC_SUCCESS;
65 }
```

## 19.5 TPM2\_SequenceComplete

### 19.5.1 General Description

This command adds the last part of data, if any, to a hash/HMAC sequence and returns the result.

NOTE 1 This command is not used to complete an Event sequence. TPM2\_EventSequenceComplete() is used for that purpose.

If for a hash sequence, the results of the hash will be used in a signing operation that uses a restricted signing key, then the ticket returned by this command can indicate that the hash is safe to sign.

If the *digest* is not safe to sign, then *validation* will be a TPMT\_TK\_HASHCHECK with the hierarchy set to TPM\_RH\_NULL and *digest* set to the Empty Buffer.

NOTE 2 Regardless of the contents of the first octets of the hashed message, if the first buffer sent to the TPM had fewer than sizeof(TPM\_GENERATED) octets, then the TPM will operate as if *digest* is not safe to sign.

If *sequenceHandle* references an Event sequence, then the TPM shall return TPM\_RC\_MODE.

Proper authorization for the sequence object associated with *sequenceHandle* is required. If an authorization or audit of this command requires computation of a *cpHash* and an *rpHash*, the Name associated with *sequenceHandle* will be the Empty Buffer.

If this command completes successfully, the *sequenceHandle* object will be flushed.

19.5.2 Command and Response

Table 73 — TPM2\_SequenceComplete Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_SequenceComplete {F}
TPMI_DH_OBJECT	@sequenceHandle	authorization for the sequence Auth Index: 1 Auth Role: USER
TPM2B_MAX_BUFFER	buffer	data to be added to the hash/HMAC
TPMI_RH_HIERARCHY+	hierarchy	hierarchy of the ticket for a hash

Table 74 — TPM2\_SequenceComplete Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DIGEST	result	the returned HMAC or digest in a sized buffer
TPMT_TK_HASHCHECK	validation	ticket indicating that the sequence of octets used to compute <i>outDigest</i> did not start with TPM_GENERATED_VALUE This is a NULL Ticket when the session is HMAC.

## 19.5.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "SequenceComplete_fp.h"
3  #include <Platform.h>

```

Error Returns	Meaning
TPM_RC_TYPE	<i>sequenceHandle</i> does not reference a hash or HMAC sequence object

```

4  TPM_RC
5  TPM2_SequenceComplete(
6      SequenceComplete_In      *in,           // IN: input parameter list
7      SequenceComplete_Out     *out          // OUT: output parameter list
8  )
9  {
10     OBJECT                      *object;
11
12     // Input validation
13
14     // Get hash object pointer
15     object = ObjectGet(in->sequenceHandle);
16
17     // input handle must be a hash or HMAC sequence object.
18     if( object->attributes.hashSeq == CLEAR
19         && object->attributes.hmacSeq == CLEAR)
20         return TPM_RC_MODE + RC_SequenceComplete_sequenceHandle;
21
22     // Command Output
23
24     if(object->attributes.hashSeq == SET)           // sequence object for hash
25     {
26         // Update last piece of data
27         HASH_OBJECT *hashObject = (HASH_OBJECT *)object;
28         CryptUpdateDigest2B(&hashObject->state.hashState[0], &in->buffer.b);
29
30         // Complete hash
31         out->result.t.size
32             = CryptGetHashDigestSize(
33                 CryptGetContextAlg(&hashObject->state.hashState[0]));
34
35         CryptCompleteHash2B(&hashObject->state.hashState[0], &out->result.b);
36
37         // Check if the first block of the sequence has been received
38         if(hashObject->attributes.firstBlock == CLEAR)
39         {
40             // If not, then this is the first block so see if it is 'safe'
41             // to sign.
42             if(TicketIsSafe(&in->buffer.b))
43                 hashObject->attributes.ticketSafe = SET;
44         }
45
46         // Output ticket
47         out->validation.tag = TPM_ST_HASHCHECK;
48         out->validation.hierarchy = in->hierarchy;
49
50         if(in->hierarchy == TPM_RH_NULL)
51         {
52             // Ticket is not required
53             out->validation.digest.t.size = 0;
54         }

```

```
55     else if(object->attributes.ticketSafe == CLEAR)
56     {
57         // Ticket is not safe to generate
58         out->validation.hierarchy = TPM_RH_NULL;
59         out->validation.digest.t.size = 0;
60     }
61     else
62     {
63         // Compute ticket
64         TicketComputeHashCheck(out->validation.hierarchy,
65                               &out->result, &out->validation);
66     }
67 }
68 else
69 {
70     HASH_OBJECT    *hashObject = (HASH_OBJECT *)object;
71
72     // Update last piece of data
73     CryptUpdateDigest2B(&hashObject->state.hmacState, &in->buffer.b);
74     // Complete hash/HMAC
75     out->result.t.size =
76         CryptGetHashDigestSize(
77             CryptGetContextAlg(&hashObject->state.hmacState.hashState));
78     CryptCompleteHMAC2B(&(hashObject->state.hmacState), &out->result.b);
79
80     // No ticket is generated for HMAC sequence
81     out->validation.tag = TPM_ST_HASHCHECK;
82     out->validation.hierarchy = TPM_RH_NULL;
83     out->validation.digest.t.size = 0;
84 }
85
86 // Internal Data Update
87
88 // mark sequence object as evict so it will be flushed on the way out
89 object->attributes.evict = SET;
90
91 return TPM_RC_SUCCESS;
92 }
```

## 19.6 TPM2\_EventSequenceComplete

### 19.6.1 General Description

This command adds the last part of data, if any, to an Event sequence and returns the result in a digest list. If *pcrHandle* references a PCR and not TPM\_RH\_NULL, then the returned digest list is processed in the same manner as the digest list input parameter to TPM2\_PCR\_Extend() with the *pcrHandle* in each bank extended with the associated digest value.

If *sequenceHandle* references a hash or HMAC sequence, the TPM shall return TPM\_RC\_MODE.

Proper authorization for the sequence object associated with *sequenceHandle* is required. If an authorization or audit of this command requires computation of a *cpHash* and an *rpHash*, the Name associated with *sequenceHandle* will be the Empty Buffer.

If this command completes successfully, the *sequenceHandle* object will be flushed.



19.6.2 Command and Response

Table 75 — TPM2\_EventSequenceComplete Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_EventSequenceComplete {NV F}
TPMI_DH_PCR+	@ pcrHandle	PCR to be extended with the Event data Auth Index: 1 Auth Role: USER
TPMI_DH_OBJECT	@sequenceHandle	authorization for the sequence Auth Index: 2 Auth Role: USER
TPM2B_MAX_BUFFER	buffer	data to be added to the Event

Table 76 — TPM2\_EventSequenceComplete Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPML_DIGEST_VALUES	results	list of digests computed for the PCR

## 19.6.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "EventSequenceComplete_fp.h"

```

Error Returns	Meaning
TPM_RC_LOCALITY	PCR extension is not allowed at the current locality
TPM_RC_MODE	input handle is not a valid event sequence object

```

3  TPM_RC
4  TPM2_EventSequenceComplete(
5      EventSequenceComplete_In      *in,          // IN: input parameter list
6      EventSequenceComplete_Out     *out         // OUT: output parameter list
7  )
8  {
9      TPM_RC      result;
10     HASH_OBJECT *hashObject;
11     UINT32      i;
12     TPM_ALG_ID  hashAlg;
13
14     // Input validation
15
16     // get the event sequence object pointer
17     hashObject = (HASH_OBJECT *)ObjectGet(in->sequenceHandle);
18
19     // input handle must reference an event sequence object
20     if(hashObject->attributes.eventSeq != SET)
21         return TPM_RC_MODE + RC_EventSequenceComplete_sequenceHandle;
22
23     // see if a PCR extend is requested in call
24     if(in->pcrHandle != TPM_RH_NULL)
25     {
26         // see if extend of the PCR is allowed at the locality of the command,
27         if(!PCRIsExtendAllowed(in->pcrHandle))
28             return TPM_RC_LOCALITY;
29         // if an extend is going to take place, then check to see if there has
30         // been an orderly shutdown. If so, and the selected PCR is one of the
31         // state saved PCR, then the orderly state has to change. The orderly state
32         // does not change for PCR that are not preserved.
33         // NOTE: This doesn't just check for Shutdown(STATE) because the orderly
34         // state will have to change if this is a state-saved PCR regardless
35         // of the current state. This is because a subsequent Shutdown(STATE) will
36         // check to see if there was an orderly shutdown and not do anything if
37         // there was. So, this must indicate that a future Shutdown(STATE) has
38         // something to do.
39         if(gp.orderlyState != SHUTDOWN_NONE && PCRIsStateSaved(in->pcrHandle))
40         {
41             result = NvIsAvailable();
42             if(result != TPM_RC_SUCCESS) return result;
43             g_clearOrderly = TRUE;
44         }
45     }
46
47     // Command Output
48
49     out->results.count = 0;
50
51     for(i = 0; i < HASH_COUNT; i++)
52     {
53         hashAlg = CryptGetHashAlgByIndex(i);

```

```
54     // Update last piece of data
55     CryptUpdateDigest2B(&hashObject->state.hashState[i], &in->buffer.b);
56     // Complete hash
57     out->results.digests[out->results.count].hashAlg = hashAlg;
58     CryptCompleteHash(&hashObject->state.hashState[i],
59                     CryptGetHashDigestSize(hashAlg),
60                     (BYTE *) &out->results.digests[out->results.count].digest);
61
62     // Extend PCR
63     if(in->pcrHandle != TPM_RH_NULL)
64         PCRExtend(in->pcrHandle, hashAlg,
65                 CryptGetHashDigestSize(hashAlg),
66                 (BYTE *) &out->results.digests[out->results.count].digest);
67     out->results.count++;
68 }
69
70 // Internal Data Update
71
72 // mark sequence object as evict so it will be flushed on the way out
73 hashObject->attributes.evict = SET;
74
75 return TPM_RC_SUCCESS;
76 }
```

## 20 Attestation Commands

### 20.1 Introduction

The attestation commands cause the TPM to sign an internally generated data structure. The contents of the data structure vary according to the command.

For all signing commands, provisions are made for the caller to provide a scheme to be used for the signing operation. This scheme will be applied only if the scheme of the key is TPM\_ALG\_NULL. If the scheme for *signHandle* is not TPM\_ALG\_NULL, then *inScheme.scheme* shall be TPM\_ALG\_NULL or the same as *scheme* in the public area of the key. If the scheme for *signHandle* is TPM\_ALG\_NULL, then *inScheme* will be used for the signing operation and may not be TPM\_ALG\_NULL. The TPM shall return TPM\_RC\_SCHEME to indicate that the scheme is not appropriate.

For a signing key that is not restricted, the caller may specify the scheme to be used as long as the scheme is compatible with the family of the key (for example, TPM\_ALG\_RSAPSS cannot be selected for an ECC key). If the caller sets *scheme* to TPM\_ALG\_NULL, then the default scheme of the key is used.

If the handle for the signing key (*signHandle*) is TPM\_RH\_NULL, then all of the actions of the command are performed and the attestation block is “signed” with the NULL Signature.

NOTE 1 This mechanism is provided so that additional commands are not required to access the data that might be in an attestation structure.

NOTE 2 When *signHandle* is TPM\_RH\_NULL, *scheme* is still required to be a valid signing scheme (may be TPM\_ALG\_NULL), but the scheme will have no effect on the format of the signature. It will always be the NULL Signature.

TPM2\_NV\_Certify() is an attestation command that is documented in 1. The remaining attestation commands are collected in the remainder of this clause.

Each of the attestation structures contains a TPMS\_CLOCK\_INFO structure and a firmware version number. These values may be considered privacy-sensitive, because they would aid in the correlation of attestations by different keys. To provide improved privacy, the *resetCount*, *restartCount*, and *firmwareVersion* numbers are obfuscated when the signing key is not in the Endorsement or Platform hierarchies.

The obfuscation value is computed by:

$$\text{obfuscation} := \text{KDFa}(\text{signHandle} \rightarrow \text{nameAlg}, \text{shProof}, \text{“OBFUSCATE”}, \text{signHandle} \rightarrow \text{QN}, 0, 128) \quad (3)$$

Of the returned 128 bits, 64 bits are added to the *versionNumber* field of the attestation structure; 32 bits are added to the *clockInfo.resetCount* and 32 bits are added to the *clockInfo.restartCount*. The order in which the bits are added is implementation-dependent.

NOTE 3 The obfuscation value for each signing key will be unique to that key in a specific location. That is, each version of a duplicated signing key will have a different obfuscation value.

When the signing key is TPM\_RH\_NULL, the data structure is produced but not signed; and the values in the signed data structure are obfuscated. When computing the obfuscation value for TPM\_RH\_NULL, the hash used for context integrity is used.

NOTE 4 The QN for TPM\_RH\_NULL is TPM\_RH\_NULL.

If the signing scheme of *signHandle* is an anonymous scheme, then the attestation blocks will not contain the Qualified Name of the *signHandle*.

Each of the attestation structures allows the caller to provide some qualifying data (*qualifyingData*). For most signing schemes, this value will be placed in the TPMS\_ATTEST.*extraData* parameter that is then

hashed and signed. However, for some schemes such as ECDA, the *qualifyingData* is used in a different manner (for details, see “ECDA” in Part 1).

## 20.2 TPM2\_Certify

### 20.2.1 General Description

The purpose of this command is to prove that an object with a specific Name is loaded in the TPM. By certifying that the object is loaded, the TPM warrants that a public area with a given Name is self-consistent and associated with a valid sensitive area. If a relying party has a public area that has the same Name as a Name certified with this command, then the values in that public area are correct.

NOTE 1 See 20.1 for description of how the signing scheme is selected.

Authorization for *objectHandle* requires ADMIN role authorization. If performed with a policy session, the session shall have a *policySession*→*commandCode* set to TPM\_CC\_Certify.

The object may be any object that is loaded with TPM2\_Load() or TPM2\_CreatePrimary(). An object that only has its public area loaded cannot be certified.

NOTE 2 The restriction occurs because the Name is used to identify the object being certified. If the TPM has not validated that the public area is associated with a matched sensitive area, then the public area may not represent a valid object and cannot be certified.

The certification includes the Name and Qualified Name of the certified object as well as the Name and the Qualified Name of the certifying object.

20.2.2 Command and Response

Table 77 — TPM2\_Certify Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Certify
TPMI_DH_OBJECT	@objectHandle	handle of the object to be certified Auth Index: 1 Auth Role: ADMIN
TPMI_DH_OBJECT+	@signHandle	handle of the key used to sign the attestation structure Auth Index: 2 Auth Role: USER
TPM2B_DATA	qualifyingData	user provided qualifying data
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the <i>scheme</i> for <i>signHandle</i> is TPM_ALG_NULL

Table 78 — TPM2\_Certify Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	.
TPM2B_ATTEST	certifyInfo	the structure that was signed
TPMT_SIGNATURE	signature	the asymmetric signature over <i>certifyInfo</i> using the key referenced by <i>signHandle</i>

## 20.2.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Attest_spt_fp.h"
3  #include "Certify_fp.h"

```

Error Returns	Meaning
TPM_RC_KEY	key referenced by <i>signHandle</i> is not a signing key
TPM_RC_SCHEME	<i>inScheme</i> is not compatible with <i>signHandle</i>
TPM_RC_VALUE	digest generated for <i>inScheme</i> is greater or has larger size than the modulus of <i>signHandle</i> , or the buffer for the result in <i>signature</i> is too small (for an RSA key); invalid commit status (for an ECC key with a split scheme).

```

4  TPM_RC
5  TPM2_Certify(
6      Certify_In      *in,                // IN: input parameter list
7      Certify_Out     *out               // OUT: output parameter list
8  )
9  {
10     TPM_RC          result;
11     TPMS_ATTEST    certifyInfo;
12
13
14     // Command Output
15
16     // Filling in attest information
17     // Common fields
18     result = FillInAttestInfo(in->signHandle,
19                             &in->inScheme,
20                             &in->qualifyingData,
21                             &certifyInfo);
22     if(result != TPM_RC_SUCCESS)
23     {
24         if(result == TPM_RC_KEY)
25             return TPM_RC_KEY + RC_Certify_signHandle;
26         else
27             return RcSafeAddToResult(result, RC_Certify_inScheme);
28     }
29     // Certify specific fields
30     // Attestation type
31     certifyInfo.type = TPM_ST_ATTEST_CERTIFY;
32     // Certified object name
33     certifyInfo.attested.certify.name.t.size =
34         ObjectGetName(in->objectHandle,
35                     certifyInfo.attested.certify.name.t.name);
36     // Certified object qualified name
37     ObjectGetQualifiedname(in->objectHandle,
38                           &certifyInfo.attested.certify.qualifiedName);
39
40     // Sign attestation structure. A NULL signature will be returned if
41     // signHandle is TPM_RH_NULL. A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
42     // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned
43     // by SignAttestInfo()
44     result = SignAttestInfo(in->signHandle,
45                             &in->inScheme,
46                             &certifyInfo,
47                             &in->qualifyingData,
48                             &out->certifyInfo,
49                             &out->signature);

```



```
50
51 // TPM_RC_ATTRIBUTES cannot be returned here as FillInAttestInfo would already
52 // have returned TPM_RC_KEY
53 pAssert(result != TPM_RC_ATTRIBUTES);
54
55 if(result != TPM_RC_SUCCESS)
56     return result;
57
58 // orderly state should be cleared because of the reporting of clock info
59 // if signing happens
60 if(in->signHandle != TPM_RH_NULL)
61     g_clearOrderly = TRUE;
62
63 return TPM_RC_SUCCESS;
64 }
```

## 20.3 TPM2\_CertifyCreation

### 20.3.1 General Description

This command is used to prove the association between an object and its creation data. The TPM will validate that the ticket was produced by the TPM and that the ticket validates the association between a loaded public area and the provided hash of the creation data (*creationHash*).

NOTE 1            See 20.1 for description of how the signing scheme is selected.

The TPM will create a test ticket using the Name associated with *objectHandle* and *creationHash* as:

$$\mathbf{HMAC}(\mathit{proof}, (\text{TPM\_ST\_CREATION} \parallel \mathit{objectHandle} \rightarrow \mathit{Name} \parallel \mathit{creationHash})) \quad (4)$$

This ticket is then compared to creation ticket. If the tickets are not the same, the TPM shall return TPM\_RC\_TICKET.

If the ticket is valid, then the TPM will create a TPMS\_ATTEST structure and place *creationHash* of the command in the *creationHash* field of the structure. The Name associated with *objectHandle* will be included in the attestation data that is then signed using the key associated with *signHandle*.

NOTE 2            If *signHandle* is TPM\_RH\_NULL, the TPMS\_ATTEST structure is returned and *signature* is a NULL Signature.

*ObjectHandle* may be any object that is loaded with TPM2\_Load() or TPM2\_CreatePrimary().

20.3.2 Command and Response

Table 79 — TPM2\_CertifyCreation Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_CertifyCreation
TPMI_DH_OBJECT+	@signHandle	handle of the key that will sign the attestation block Auth Index: 1 Auth Role: USER
TPMI_DH_OBJECT	objectHandle	the object associated with the creation data Auth Index: None
TPM2B_DATA	qualifyingData	user-provided qualifying data
TPM2B_DIGEST	creationHash	hash of the creation data produced by TPM2_Create() or TPM2_CreatePrimary()
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the <i>scheme</i> for <i>signHandle</i> is TPM_ALG_NULL
TPMT_TK_CREATION	creationTicket	ticket produced by TPM2_Create() or TPM2_CreatePrimary()

Table 80 — TPM2\_CertifyCreation Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ATTEST	certifyInfo	the structure that was signed
TPMT_SIGNATURE	signature	the signature over <i>certifyInfo</i>

## 20.3.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Attest_spt_fp.h"
3  #include "CertifyCreation_fp.h"

```

Error Returns	Meaning
TPM_RC_KEY	key referenced by <i>signHandle</i> is not a signing key
TPM_RC_SCHEME	<i>inScheme</i> is not compatible with <i>signHandle</i>
TPM_RC_TICKET	<i>creationTicket</i> does not match <i>objectHandle</i>
TPM_RC_VALUE	digest generated for <i>inScheme</i> is greater or has larger size than the modulus of <i>signHandle</i> , or the buffer for the result in <i>signature</i> is too small (for an RSA key); invalid commit status (for an ECC key with a split scheme).

```

4  TPM_RC
5  TPM2_CertifyCreation(
6      CertifyCreation_In    *in,           // IN: input parameter list
7      CertifyCreation_Out  *out          // OUT: output parameter list
8  )
9  {
10     TPM_RC                result;
11     TPM2B_NAME            name;
12     TPMT_TK_CREATION      ticket;
13     TPMS_ATTEST           certifyInfo;
14
15     // Input Validation
16
17     // CertifyCreation specific input validation
18     // Get certified object name
19     name.t.size = ObjectGetName(in->objectHandle, name.t.name);
20     // Re-compute ticket
21     TicketComputeCreation(in->creationTicket.hierarchy, &name,
22                          &in->creationHash, &ticket);
23     // Compare ticket
24     if(!Memory2BEqual(&ticket.digest.b, &in->creationTicket.digest.b))
25         return TPM_RC_TICKET + RC_CertifyCreation_creationTicket;
26
27     // Command Output
28     // Common fields
29     result = FillInAttestInfo(in->signHandle, &in->inScheme, &in->qualifyingData,
30                                &certifyInfo);
31     if(result != TPM_RC_SUCCESS)
32     {
33         if(result == TPM_RC_KEY)
34             return TPM_RC_KEY + RC_CertifyCreation_signHandle;
35         else
36             return RcSafeAddToResult(result, RC_CertifyCreation_inScheme);
37     }
38
39     // CertifyCreation specific fields
40     // Attestation type
41     certifyInfo.type = TPM_ST_ATTEST_CREATION;
42     certifyInfo.attested.creation.objectName = name;
43
44     // Copy the creationHash
45     certifyInfo.attested.creation.creationHash = in->creationHash;
46
47     // Sign attestation structure. A NULL signature will be returned if

```

```
48     // signHandle is TPM_RH_NULL. A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
49     // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned at
50     // this point
51     result = SignAttestInfo(in->signHandle,
52                           &in->inScheme,
53                           &certifyInfo,
54                           &in->qualifyingData,
55                           &out->certifyInfo,
56                           &out->signature);
57
58     // TPM_RC_ATTRIBUTES cannot be returned here as FillInAttestInfo would already
59     // have returned TPM_RC_KEY
60     pAssert(result != TPM_RC_ATTRIBUTES);
61
62     if(result != TPM_RC_SUCCESS)
63         return result;
64
65     // orderly state should be cleared because of the reporting of clock info
66     // if signing happens
67     if(in->signHandle != TPM_RH_NULL)
68         g_clearOrderly = TRUE;
69
70     return TPM_RC_SUCCESS;
71 }
```

## 20.4 TPM2\_Quote

### 20.4.1 General Description

This command is used to quote PCR values.

NOTE See 20.1 for description of how the signing scheme is selected.

The TPM will hash the list of PCR selected by *PCRselect* using the hash algorithm associated with *signHandle* (this is the hash algorithm of the signing scheme, not the *nameAlg* of *signHandle*).

The digest is computed as the hash of the concatenation of all of the digest values of the selected PCR.

The concatenation of PCR is described in Part 1, *Selecting Multiple PCR*.

20.4.2 Command and Response

Table 81 — TPM2\_Quote Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Quote
TPMI_DH_OBJECT	@signHandle	handle of key that will perform signature Auth Index: 1 Auth Role: USER
TPM2B_DATA	qualifyingData	data supplied by the caller
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the <i>scheme</i> for <i>signHandle</i> is TPM_ALG_NULL
TPML_PCR_SELECTION	PCRselect	PCR set to quote

Table 82 — TPM2\_Quote Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ATTEST	quoted	the quoted information
TPMT_SIGNATURE	signature	the signature over <i>quoted</i>

## 20.4.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Attest_spt_fp.h"
3  #include "Quote_fp.h"

```

Error Returns	Meaning
TPM_RC_KEY	<i>signHandle</i> does not reference a signing key;
TPM_RC_SCHEME	the scheme is not compatible with sign key type, or input scheme is not compatible with default scheme, or the chosen scheme is not a valid sign scheme

```

4  TPM_RC
5  TPM2_Quote(
6      Quote_In          *in,           // IN: input parameter list
7      Quote_Out         *out          // OUT: output parameter list
8  )
9  {
10     TPM_RC              result;
11     TPMS_ALG_HASH      hashAlg;
12     TPMS_ATTEST        quoted;
13
14     // Command Output
15
16     // Filling in attest information
17     // Common fields
18     // FillInAttestInfo will return TPM_RC_SCHEME or TPM_RC_KEY
19     result = FillInAttestInfo(in->signHandle,
20                             &in->inScheme,
21                             &in->qualifyingData,
22                             &quoted);
23     if(result != TPM_RC_SUCCESS)
24     {
25         if(result == TPM_RC_KEY)
26             return TPM_RC_KEY + RC_Quote_signHandle;
27         else
28             return RcSafeAddToResult(result, RC_Quote_inScheme);
29     }
30
31     // Quote specific fields
32     // Attestation type
33     quoted.type = TPM_ST_ATTEST_QUOTE;
34
35     // Get hash algorithm in sign scheme. This hash algorithm is used to
36     // compute PCR digest. If there is no algorithm, then the PCR cannot
37     // be digested and this command returns TPM_RC_SCHEME
38     hashAlg = in->inScheme.details.any.hashAlg;
39
40     if(hashAlg == TPM_ALG_NULL)
41         return TPM_RC_SCHEME + RC_Quote_inScheme;
42
43     // Compute PCR digest
44     PCRComputeCurrentDigest(hashAlg,
45                             &in->PCRselect,
46                             &quoted.attested.quote.pcrDigest);
47
48     // Copy PCR select. "PCRselect" is modified in PCRComputeCurrentDigest
49     // function
50     quoted.attested.quote.pcrSelect = in->PCRselect;
51
52     // Sign attestation structure. A NULL signature will be returned if

```



```
53 // signHandle is TPM_RH_NULL. TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES
54 // error may be returned by SignAttestInfo.
55 // NOTE: TPM_RC_ATTRIBUTES means that the key is not a signing key but that
56 // was checked above and TPM_RC_KEY was returned. TPM_RC_VALUE means that the
57 // value to sign is too large but that means that the digest is too big and
58 // that can't happen.
59 result = SignAttestInfo(in->signHandle,
60                        &in->inScheme,
61                        &quoted,
62                        &in->qualifyingData,
63                        &out->quoted,
64                        &out->signature);
65 if(result != TPM_RC_SUCCESS)
66     return result;
67
68 // orderly state should be cleared because of the reporting of clock info
69 // if signing happens
70 if(in->signHandle != TPM_RH_NULL)
71     g_clearOrderly = TRUE;
72
73 return TPM_RC_SUCCESS;
74 }
```

## 20.5 TPM2\_GetSessionAuditDigest

### 20.5.1 General Description

This command returns a digital signature of the audit session digest.

NOTE 1 See 20.1 for description of how the signing scheme is selected.

If *sessionHandle* is not an audit session, the TPM shall return TPM\_RC\_TYPE.

NOTE 2 A session does not become an audit session until the successful completion of the command in which the session is first used as an audit session.

This command requires authorization from the privacy administrator of the TPM (expressed with *endorsementAuth*) as well as authorization to use the key associated with *signHandle*.

If this command is audited, then the audit digest that is signed will not include the digest of this command because the audit digest is only updated when the command completes successfully.

This command does not cause the audit session to be closed and does not reset the digest value.

NOTE 3 The audit session digest will be reset if the *sessionHandle* is used as the audit session for the command and the *auditReset* attribute of the session is set; and this command will be the first command in the audit digest.

NOTE 4 A reason for using 'sessionHandle' in this command is so that the *continueSession* attribute may be CLEAR. This will flush the session at the end of the command.

## 20.5.2 Command and Response

Table 83 — TPM2\_GetSessionAuditDigest Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_GetSessionAuditDigest
TPMI_RH_ENDORSEMENT	@privacyAdminHandle	handle of the privacy administrator (TPM_RH_ENDORSEMENT) Auth Index: 1 Auth Role: USER
TPMI_DH_OBJECT+	@signHandle	handle of the signing key Auth Index: 2 Auth Role: USER
TPMI_SH_HMAC	sessionHandle	handle of the audit session Auth Index: None
TPM2B_DATA	qualifyingData	user-provided qualifying data – may be zero-length
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the <i>scheme</i> for <i>signHandle</i> is TPM_ALG_NULL

Table 84 — TPM2\_GetSessionAuditDigest Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ATTEST	auditInfo	the audit information that was signed
TPMT_SIGNATURE	signature	the signature over <i>auditInfo</i>

## 20.5.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Attest_spt_fp.h"
3  #include "GetSessionAuditDigest_fp.h"

```

Error Returns	Meaning
TPM_RC_KEY	key referenced by <i>signHandle</i> is not a signing key
TPM_RC_SCHEME	<i>inScheme</i> is incompatible with <i>signHandle</i> type; or both <i>scheme</i> and key's default scheme are empty; or <i>scheme</i> is empty while key's default scheme requires explicit input scheme (split signing); or non-empty default key scheme differs from <i>scheme</i>
TPM_RC_TYPE	<i>sessionHandle</i> does not reference an audit session
TPM_RC_VALUE	digest generated for the given <i>scheme</i> is greater than the modulus of <i>signHandle</i> (for an RSA key); invalid commit status or failed to generate r value (for an ECC key)

```

4  TPM_RC
5  TPM2_GetSessionAuditDigest(
6      GetSessionAuditDigest_In      *in,           // IN: input parameter list
7      GetSessionAuditDigest_Out     *out          // OUT: output parameter list
8  )
9  {
10     TPM_RC          result;
11     SESSION        *session;
12     TPMS_ATTEST    auditInfo;
13
14     // Input Validation
15
16     // SessionAuditDigest specific input validation
17     // Get session pointer
18     session = SessionGet(in->sessionHandle);
19
20     // session must be an audit session
21     if(session->attributes.isAudit == CLEAR)
22         return TPM_RC_TYPE + RC_GetSessionAuditDigest_sessionHandle;
23
24     // Command Output
25
26     // Filling in attest information
27     // Common fields
28     result = FillInAttestInfo(in->signHandle,
29                             &in->inScheme,
30                             &in->qualifyingData,
31                             &auditInfo);
32     if(result != TPM_RC_SUCCESS)
33     {
34         if(result == TPM_RC_KEY)
35             return TPM_RC_KEY + RC_GetSessionAuditDigest_signHandle;
36         else
37             return RcSafeAddToResult(result, RC_GetSessionAuditDigest_inScheme);
38     }
39
40     // SessionAuditDigest specific fields
41     // Attestation type
42     auditInfo.type = TPM_ST_ATTEST_SESSION_AUDIT;
43
44     // Copy digest

```

```
45     auditInfo.attested.sessionAudit.sessionDigest = session->u2.auditDigest;
46
47     // Exclusive audit session
48     if(g_exclusiveAuditSession == in->sessionHandle)
49         auditInfo.attested.sessionAudit.exclusiveSession = TRUE;
50     else
51         auditInfo.attested.sessionAudit.exclusiveSession = FALSE;
52
53     // Sign attestation structure. A NULL signature will be returned if
54     // signHandle is TPM_RH_NULL. A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
55     // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned at
56     // this point
57     result = SignAttestInfo(in->signHandle,
58                             &in->inScheme,
59                             &auditInfo,
60                             &in->qualifyingData,
61                             &out->auditInfo,
62                             &out->signature);
63     if(result != TPM_RC_SUCCESS)
64         return result;
65
66     // orderly state should be cleared because of the reporting of clock info
67     // if signing happens
68     if(in->signHandle != TPM_RH_NULL)
69         g_clearOrderly = TRUE;
70
71     return TPM_RC_SUCCESS;
72 }
```

## 20.6 TPM2\_GetCommandAuditDigest

### 20.6.1 General Description

This command returns the current value of the command audit digest, a digest of the commands being audited, and the audit hash algorithm. These values are placed in an attestation structure and signed with the key referenced by *signHandle*.

NOTE 1            See 20.1 for description of how the signing scheme is selected.

When this command completes successfully, and *signHandle* is not TPM\_RH\_NULL, the audit digest is cleared.

NOTE 2            The way that the TPM tracks that the digest is clear is vendor-dependent. The reference implementation resets the size of the digest to zero.

If this command is being audited, then the signed digest produced by the command will not include the command. At the end of this command, the audit digest will be extended with *cpHash* and the *rpHash* of the command which would change the command audit digest signed by the next invocation of this command.

This command requires authorization from the privacy administrator of the TPM (expressed with *endorsementAuth*) as well as authorization to use the key associated with *signHandle*.

20.6.2 Command and Response

**Table 85 — TPM2\_GetCommandAuditDigest Command**

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_GetCommandAuditDigest {NV}
TPMI_RH_ENDORSEMENT	@privacyHandle	handle of the privacy administrator (TPM_RH_ENDORSEMENT) Auth Index: 1 Auth Role: USER
TPMI_DH_OBJECT+	@signHandle	the handle of the signing key Auth Index: 2 Auth Role: USER
TPM2B_DATA	qualifyingData	other data to associate with this audit digest
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the <i>scheme</i> for <i>signHandle</i> is TPM_ALG_NULL

**Table 86 — TPM2\_GetCommandAuditDigest Response**

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ATTEST	auditInfo	the auditInfo that was signed
TPMT_SIGNATURE	signature	the signature over <i>auditInfo</i>

## 20.6.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Attest_spt_fp.h"
3  #include "GetCommandAuditDigest_fp.h"

```

Error Returns	Meaning
TPM_RC_KEY	key referenced by <i>signHandle</i> is not a signing key
TPM_RC_SCHEME	<i>inScheme</i> is incompatible with <i>signHandle</i> type; or both <i>scheme</i> and key's default scheme are empty; or <i>scheme</i> is empty while key's default scheme requires explicit input scheme (split signing); or non-empty default key scheme differs from <i>scheme</i>
TPM_RC_VALUE	digest generated for the given <i>scheme</i> is greater than the modulus of <i>signHandle</i> (for an RSA key); invalid commit status or failed to generate r value (for an ECC key)

```

4  TPM_RC
5  TPM2_GetCommandAuditDigest(
6      GetCommandAuditDigest_In    *in,           // IN: input parameter list
7      GetCommandAuditDigest_Out   *out          // OUT: output parameter list
8  )
9  {
10     TPM_RC          result;
11     TPMS_ATTEST    auditInfo;
12
13     // Command Output
14
15     // Filling in attest information
16     // Common fields
17     result = FillInAttestInfo(in->signHandle,
18                             &in->inScheme,
19                             &in->qualifyingData,
20                             &auditInfo);
21     if(result != TPM_RC_SUCCESS)
22     {
23         if(result == TPM_RC_KEY)
24             return TPM_RC_KEY + RC_GetCommandAuditDigest_signHandle;
25         else
26             return RcSafeAddToResult(result, RC_GetCommandAuditDigest_inScheme);
27     }
28
29     // CommandAuditDigest specific fields
30     // Attestation type
31     auditInfo.type = TPM_ST_ATTEST_COMMAND_AUDIT;
32
33     // Copy audit hash algorithm
34     auditInfo.attested.commandAudit.digestAlg = gp.auditHashAlg;
35
36     // Copy counter value
37     auditInfo.attested.commandAudit.auditCounter = gp.auditCounter;
38
39     // Copy command audit log
40     auditInfo.attested.commandAudit.auditDigest = gr.commandAuditDigest;
41     CommandAuditGetDigest(&auditInfo.attested.commandAudit.commandDigest);
42
43     // Sign attestation structure. A NULL signature will be returned if
44     // signHandle is TPM_RH_NULL. A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
45     // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned at
46     // this point

```



```
47     result = SignAttestInfo(in->signHandle,
48                             &in->inScheme,
49                             &auditInfo,
50                             &in->qualifyingData,
51                             &out->auditInfo,
52                             &out->signature);
53
54     if(result != TPM_RC_SUCCESS)
55         return result;
56
57     // Internal Data Update
58
59     if(in->signHandle != TPM_RH_NULL)
60     {
61         // Reset log
62         gr.commandAuditDigest.t.size = 0;
63
64         // orderly state should be cleared because of the update in
65         // commandAuditDigest, as well as the reporting of clock info
66         g_clearOrderly = TRUE;
67     }
68
69     return TPM_RC_SUCCESS;
70 }
```

## 20.7 TPM2\_GetTime

### 20.7.1 General Description

This command returns the current values of *Time* and *Clock*.

NOTE 1 See 20.1 for description of how the signing scheme is selected.

The values of *Clock*, *resetCount* and *restartCount* appear in two places in *timeInfo*: once in `TPMS_ATTEST.clockInfo` and again in `TPMS_ATTEST.attested.time.clockInfo`. The firmware version number also appears in two places (`TPMS_ATTEST.firmwareVersion` and `TPMS_ATTEST.attested.time.firmwareVersion`). If *signHandle* is in the endorsement or platform hierarchies, both copies of the data will be the same. However, if *signHandle* is in the storage hierarchy or is `TPM_RH_NULL`, the values in `TPMS_ATTEST.clockInfo` and `TPMS_ATTEST.firmwareVersion` are obfuscated but the values in `TPM_ATTEST.attested.time` are not.

NOTE 2 The purpose of this duplication is to allow an entity who is trusted by the privacy Administrator to correlate the obfuscated values with the clear-text values.

20.7.2 Command and Response

Table 87 — TPM2\_GetTime Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_GetTime
TPMI_RH_ENDORSEMENT	@privacyAdminHandle	handle of the privacy administrator (TPM_RH_ENDORSEMENT) Auth Index: 1 Auth Role: USER
TPMI_DH_OBJECT+	@signHandle	the <i>keyHandle</i> identifier of a loaded key that can perform digital signatures Auth Index: 2 Auth Role: USER
TPM2B_DATA	qualifyingData	data to tick stamp
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the <i>scheme</i> for <i>signHandle</i> is TPM_ALG_NULL

Table 88 — TPM2\_GetTime Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	.
TPM2B_ATTEST	timeInfo	standard TPM-generated attestation block
TPMT_SIGNATURE	signature	the signature over <i>timeInfo</i>

## 20.7.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Attest_spt_fp.h"
3  #include "GetTime_fp.h"

```

Error Returns	Meaning
TPM_RC_KEY	key referenced by <i>signHandle</i> is not a signing key
TPM_RC_SCHEME	<i>inScheme</i> is incompatible with <i>signHandle</i> type; or both <i>scheme</i> and key's default scheme are empty; or <i>scheme</i> is empty while key's default scheme requires explicit input scheme (split signing); or non-empty default key scheme differs from <i>scheme</i>
TPM_RC_VALUE	digest generated for the given <i>scheme</i> is greater than the modulus of <i>signHandle</i> (for an RSA key); invalid commit status or failed to generate r value (for an ECC key)

```

4  TPM_RC
5  TPM2_GetTime (
6      GetTime_In      *in,                // IN: input parameter list
7      GetTime_Out     *out               // OUT: output parameter list
8  )
9  {
10     TPM_RC          result;
11     TPMS_ATTEST     timeInfo;
12
13     // Command Output
14
15     // Filling in attest information
16     // Common fields
17     result = FillInAttestInfo(in->signHandle,
18                             &in->inScheme,
19                             &in->qualifyingData,
20                             &timeInfo);
21     if(result != TPM_RC_SUCCESS)
22     {
23         if(result == TPM_RC_KEY)
24             return TPM_RC_KEY + RC_GetTime_signHandle;
25         else
26             return RcSafeAddToResult(result, RC_GetTime_inScheme);
27     }
28
29     // GetClock specific fields
30     // Attestation type
31     timeInfo.type = TPM_ST_ATTEST_TIME;
32
33     // current clock in plain text
34     timeInfo.attested.time.time.time = g_time;
35     TimeFillInfo(&timeInfo.attested.time.time.clockInfo);
36
37     // Firmware version in plain text
38     timeInfo.attested.time.firmwareVersion
39         = ((UINT64) gp.firmwareV1) << 32;
40     timeInfo.attested.time.firmwareVersion += gp.firmwareV2;
41
42     // Sign attestation structure. A NULL signature will be returned if
43     // signHandle is TPM_RH_NULL. A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
44     // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned at
45     // this point
46     result = SignAttestInfo(in->signHandle,

```

```
47         &in->inScheme,  
48         &timeInfo,  
49         &in->qualifyingData,  
50         &out->timeInfo,  
51         &out->signature);  
52     if(result != TPM_RC_SUCCESS)  
53         return result;  
54  
55     // orderly state should be cleared because of the reporting of clock info  
56     // if signing happens  
57     if(in->signHandle != TPM_RH_NULL)  
58         g_clearOrderly = TRUE;  
59  
60     return TPM_RC_SUCCESS;  
61 }
```

## 21 Ephemeral EC Keys

### 21.1 Introduction

The TPM generates keys that have different lifetimes. TPM keys in a hierarchy can be persistent for as long as the seed of the hierarchy is unchanged and these keys may be used multiple times. Other TPM-generated keys are only useful for a single operation. Some of these single-use keys are used in the command in which they are created. Examples of this use are TPM2\_Duplicate() where an ephemeral key is created for a single pass key exchange with another TPM. However, there are other cases, such as anonymous attestation, where the protocol requires two passes where the public part of the ephemeral key is used outside of the TPM before the final command "consumes" the ephemeral key.

For these uses, TPM2\_Commit() or TPM2\_EC\_Ephemeral() may be used to have the TPM create an ephemeral EC key and return the public part of the key for external use. Then in a subsequent command, the caller provides a reference to the ephemeral key so that the TPM can retrieve or recreate the associated private key.

When an ephemeral EC key is created, it is assigned a number and that number is returned to the caller as the identifier for the key. This number is not a handle. A handle is assigned to a key that may be context saved but these ephemeral EC keys may not be saved and do not have a full key context. When a subsequent command uses the ephemeral key, the caller provides the number of the ephemeral key. The TPM uses that number to either look up or recompute the associated private key. After the key is used, the TPM records the fact that the key has been used so that it cannot be used again.

As mentioned, the TPM can keep each assigned private ephemeral key in memory until it is used. However, this could consume a large amount of memory. To limit the memory size, the TPM is allowed to restrict the number of pending private keys – keys that have been allocated but not used.

NOTE            The minimum number of ephemeral keys is determined by a platform specific specification

To further reduce the memory requirements for the ephemeral private keys, the TPM is allowed to use pseudo-random values for the ephemeral keys. Instead of keeping the full value of the key in memory, the TPM can use a counter as input to a KDF. Incrementing the counter will cause the TPM to generate a new pseudo-random value.

Using the counter to generate pseudo-random private ephemeral keys greatly simplifies tracking of key usage. When a counter value is used to create a key, a bit in an array may be set to indicate that the key use is pending. When the ephemeral key is consumed, the bit is cleared. This prevents the key from being used more than once.

Since the TPM is allowed to restrict the number of pending ephemeral keys, the array size can be limited. For example, a 128 bit array would allow 128 keys to be "pending".

The management of the array is described in greater detail in the *Split Operations* clause in Annex C of part 1.

## 21.2 TPM2\_Commit

### 21.2.1 General Description

TPM2\_Commit() performs the first part of an ECC anonymous signing operation. The TPM will perform the point multiplications on the provided points and return intermediate signing values. The *signHandle* parameter shall refer to an ECC key with the sign attribute (TPM\_RC\_ATTRIBUTES) using an anonymous signing scheme (TPM\_RC\_SCHEME).

For this command, *p1*, *s2* and *y2* are optional parameters. If *s2* is an Empty Buffer, then the TPM shall return TPM\_RC\_SIZE if *y2* is not an Empty Buffer. If *p1*, *s2*, and *y2* are all Empty Buffers, the TPM shall return TPM\_RC\_NO\_RESULT.

In the algorithm below, the following additional values are used in addition to the command parameters:

$H_{nameAlg}$	hash function using the <i>nameAlg</i> of the key associated with <i>signHandle</i>
$p$	field modulus of the curve associated with <i>signHandle</i>
$n$	order of the curve associated with <i>signHandle</i>
$d_s$	private key associated with <i>signHandle</i>
$c$	counter that increments each time a TPM2_Commit() is successfully completed
$A[i]$	array of bits used to indicate when a value of $c$ has been used in a signing operation; values of $i$ are 0 to $2n-1$
$k$	nonce that is set to a random value on each TPM Reset; nonce size is twice the security strength of any ECDSA key supported by the TPM.

The algorithm is:

- a) set  $K$ ,  $L$ , and  $E$  to be Empty Buffers.
- b) if  $s2$  is not an Empty Buffer, compute  $x2 := H_{nameAlg}(s2) \bmod p$ , else skip to step (e)
- c) if  $(x2, y2)$  is not a point on the curve of *signHandle*, return TPM\_RC\_ECC\_POINT
- d) set  $K := [d_s](x2, y2)$
- e) generate or derive  $r$  (see the "Commit Random Value" clause in Part 1)
- f) set  $r := r \bmod n$

NOTE 1  $nLen$  is the number of bits in  $n$

- g) if  $p1$  is an Empty Buffer, skip to step i)
- h) if  $(p1)$  is not a point on the curve of *signHandle*, return TPM\_RC\_ECC\_POINT
- i) set  $E := [r](p1)$
- j) if  $K$  is not an Empty Buffer, set  $L := [r](x2, y2)$
- k) if  $K$ ,  $L$ , or  $E$  is the point at infinity, return TPM\_RC\_NO\_RESULT
- l) set  $commitCount := commitCount$
- m) set  $commitCount := commitCount + 1$

NOTE 2            Depending on the method of generating *r*, it may be necessary to update the tracking array here.

n) output *K, L, E* and *counter*

NOTE 3            Depending on the input parameters *K* and *L* may be Empty Buffers or *E* may be an Empty Buffer



## 21.2.2 Command and Response

Table 89 — TPM2\_Commit Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	paramSize	
TPM_CC	commandCode	TPM_CC_Commit
TPMI_DH_OBJECT	@signHandle	handle of the key that will be used in the signing operation Auth Index: 1 Auth Role: USER
TPM2B_ECC_POINT	P1	a point ( $M$ ) on the curve used by <i>signHandle</i>
TPM2B_SENSITIVE_DATA	s2	octet array used to derive x-coordinate of a base point
TPM2B_ECC_PARAMETER	y2	y coordinate of the point associated with s2

Table 90 — TPM2\_Commit Response

Type	Name	Description
TPM_ST	tag	see 8
UINT32	paramSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	K	ECC point $K := [d_s](x_2, y_2)$
TPM2B_ECC_POINT	L	ECC point $L := [r](x_2, y_2)$
TPM2B_ECC_POINT	E	ECC point $E := [r]P_1$
UINT16	counter	least-significant 16 bits of <i>commitCount</i>

## 21.2.3 Detailed Actions

```

1  /*(Copyright)
2      Microsoft Copyright 2009, 2010, 2011, 2012, 2013
3          Microsoft Confidential Contribution to a TCG Specification or Design Guide
4          under Article 15 of "The Bylaws of the Trusted Computing Group" as Amended
5          through March 20, 2003
6
7  */
8
9  #include "InternalRoutines.h"
10 #include "Commit_fp.h"
11
12 #ifndef TPM_ALG_ECC
13
14 /*(See part 3 specification)

```

This command performs the point multiply operations for anonymous signing schemes.

```

15 */

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	<i>keyHandle</i> references a restricted key that is not a signing key
TPM_RC_ECC_POINT	either <i>P1</i> or the point derived from <i>s2</i> is not on the curve of <i>keyHandle</i>
TPM_RC_HASH	invalid name algorithm in <i>keyHandle</i>
TPM_RC_KEY	<i>keyHandle</i> does not reference an ECC key
TPM_RC_SCHEME	<i>keyHandle</i> references a restricted signing key that does not use and anonymous scheme
TPM_RC_NO_RESULT	<i>K</i> , <i>L</i> or <i>E</i> was a point at infinity; or failed to generate <i>r</i> value
TPM_RC_SIZE	<i>s2</i> is empty but <i>y2</i> is not or <i>s2</i> provided but <i>y2</i> is not

```

16 TPM_RC
17 TPM2_Commit(
18     Commit_In     *in,           // IN: input parameter list
19     Commit_Out    *out          // OUT: output parameter list
20 )
21 {
22     OBJECT                *eccKey;
23     TPMS_ECC_POINT        P2;
24     TPMS_ECC_POINT        *pP2 = NULL;
25     TPMS_ECC_POINT        *pP1 = NULL;
26     TPM2B_ECC_PARAMETER    r;
27     TPM2B                  *p;
28     TPM_RC                result;
29     UINT16                 hashResults;
30
31     // Input Validation
32
33     eccKey = ObjectGet(in->signHandle);
34
35     // Input key must be an ECC key
36     if(eccKey->publicArea.type != TPM_ALG_ECC)
37         return TPM_RC_KEY + RC_Commit_signHandle;
38
39     // if the key is restricted, it must be a signing key using an anonymous scheme
40     if(eccKey->publicArea.objectAttributes.restricted == SET)

```

```

41     {
42         if(eccKey->publicArea.objectAttributes.sign != SET)
43             return TPM_RC_ATTRIBUTES + RC_Commit_signHandle;
44         if(!CryptIsSchemeAnonymous(
45             eccKey->publicArea.parameters.eccDetail.scheme.scheme))
46             return TPM_RC_SCHEME + RC_Commit_signHandle;
47     }
48     else
49     {
50         // if not restricted, s2, and y2 must be an Empty Buffer
51         if(in->s2.t.size)
52             return TPM_RC_SIZE + RC_Commit_s2;
53     }
54     // Make sure that both parts of P2 are present if either is present
55     if((in->s2.t.size == 0) != (in->y2.t.size == 0))
56         return TPM_RC_SIZE + RC_Commit_y2;
57
58     // Get prime modulus for the curve. This is needed later but getting this now
59     // allows confirmation that the curve exists
60     p = (TPM2B *)CryptEccGetParameter('p', eccKey->publicArea.parameters.eccDetail.curveID);
61
62     // if no p, then the curve ID is bad
63     // NOTE: This should never occur if the input unmarshaling code is working
64     // correctly
65     if(p == NULL)
66         return TPM_RC_KEY + RC_Commit_signHandle;
67
68     // Get the random value that will be used in the point multiplications
69     // Note: this does not commit the count.
70     if(!CryptGenerateR(&r,
71         NULL,
72         eccKey->publicArea.parameters.eccDetail.curveID,
73         &eccKey->name))
74         return TPM_RC_NO_RESULT;
75
76     // Set up P2 if s2 and Y2 are provided
77     if(in->s2.t.size != 0)
78     {
79         pP2 = &P2;
80
81         // copy y2 for P2
82         MemoryCopy2B(&P2.y.b, &in->y2.b);
83         // Compute x2 := HnameAlg(s2) mod p
84
85         // do the hash operation on s2 with the size of curve 'p'
86         hashResults = CryptHashBlock(eccKey->publicArea.nameAlg,
87             in->s2.t.size,
88             in->s2.t.buffer,
89             p->size,
90             P2.x.t.buffer);
91
92         // If there were error returns in the hash routine, indicate a problem
93         // with the hash in
94         if(hashResults == 0)
95             return TPM_RC_HASH + RC_Commit_signHandle;
96
97         // set the size of the X value to the size of the hash
98         P2.x.t.size = hashResults;
99
100        // set p2.x = hash(s2) mod p
101        if(CryptDivide(&P2.x.b, p, NULL, &P2.x.b) != TPM_RC_SUCCESS)
102            return TPM_RC_NO_RESULT;
103
104        if(!CryptEccIsPointOnCurve(eccKey->publicArea.parameters.eccDetail.curveID,

```

```

105         pP2))
106     return TPM_RC_ECC_POINT + RC_Commit_s2;
107
108     if(eccKey->attributes.publicOnly == SET)
109         return TPM_RC_KEY + RC_Commit_signHandle;
110
111 }
112 else
113
114 // If there is a P1, make sure that it is on the curve
115 // NOTE: an "empty" point has two UINT16 values which are the size values
116 // for each of the coordinates.
117 if(in->P1.t.size > 4)
118 {
119     pP1 = &in->P1.t.point;
120     if(!CryptEccIsPointOnCurve(eccKey->publicArea.parameters.eccDetail.curveID,
121                               pP1))
122         return TPM_RC_ECC_POINT + RC_Commit_P1;
123 }
124
125 // Pass the parameters to CryptCommit.
126 // The work is not done inline because it does several point multiplies
127 // with the same curve. There is significant optimization by not
128 // having to reload the curve parameters multiple times.
129 result = CryptCommitCompute(&out->K.t.point,
130                             &out->L.t.point,
131                             &out->E.t.point,
132                             eccKey->publicArea.parameters.eccDetail.curveID,
133                             pP1,
134                             pP2,
135                             &eccKey->sensitive.sensitive.ecc,
136                             &r);
137 if(result != TPM_RC_SUCCESS)
138     return result;
139
140 out->K.t.size = TPMS_ECC_POINT_Marshal(&out->K.t.point, NULL, NULL);
141 out->L.t.size = TPMS_ECC_POINT_Marshal(&out->L.t.point, NULL, NULL);
142 out->E.t.size = TPMS_ECC_POINT_Marshal(&out->E.t.point, NULL, NULL);
143
144 // The commit computation was successful so complete the commit by setting
145 // the bit
146 out->counter = CryptCommit();
147
148 return TPM_RC_SUCCESS;
149 }
150 #endif

```

## 21.3 TPM2\_EC\_Ephemeral

### 21.3.1 General Description

TPM2\_EC\_Ephemeral() creates an ephemeral key for use in a two-phase key exchange protocol.

The TPM will use the commit mechanism to assign an ephemeral key  $r$  and compute a public point  $Q := [r]G$  where  $G$  is the generator point associated with *curveID*.

## 21.3.2 Command and Response

Table 91 — TPM2\_EC\_Ephemeral Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	paramSize	
TPM_CC	commandCode	TPM_CC_EC_Ephemeral
TPMI_ECC_CURVE	curveID	The curve for the computed ephemeral point

Table 92 — TPM2\_EC\_Ephemeral Response

Type	Name	Description
TPM_ST	tag	see 8
UINT32	paramSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	Q	ephemeral public key $Q := [r]G$
UINT16	counter	least-significant 16 bits of <i>commitCount</i>

## 21.3.3 Detailed Actions

```

1  /*(Copyright)
2      Microsoft Copyright 2009, 2010, 2011, 2012, 2013
3          Microsoft Confidential Contribution to a TCG Specification or Design Guide
4          under Article 15 of "The Bylaws of the Trusted Computing Group" as Amended
5          through March 20, 2003
6
7  */
8
9  #include "InternalRoutines.h"
10 #include "EC_Ephemeral_fp.h"
11
12 #ifndef TPM_ALG_ECC
13
14 /*(See part 3 specification)

```

This command creates an ephemeral key using the commit mechanism

```

15 */

```

Error Returns	Meaning
---------------	---------

```

16 TPM_RC
17 TPM2_EC_Ephemeral(
18     EC_Ephemeral_In      *in,           // IN: input parameter list
19     EC_Ephemeral_Out     *out          // OUT: output parameter list
20 )
21 {
22     TPM2B_ECC_PARAMETER  r;
23
24     // Get the random value that will be used in the point multiplications
25     // Note: this does not commit the count.
26     if(!CryptGenerateR(&r,
27                        NULL,
28                        in->curveID,
29                        NULL))
30         return TPM_RC_NO_RESULT;
31
32     CryptEccPointMultiply(&out->Q.t.point, in->curveID, &r, NULL);
33
34     // commit the count value
35     out->counter = CryptCommit();
36
37     return TPM_RC_SUCCESS;
38 }
39 #endif

```

## 22 Signing and Signature Verification

### 22.1 TPM2\_VerifySignature

#### 22.1.1 General Description

This command uses loaded keys to validate a signature on a message with the message digest passed to the TPM.

If the signature check succeeds, then the TPM will produce a TPMT\_TK\_VERIFIED. Otherwise, the TPM shall return TPM\_RC\_SIGNATURE.

NOTE 1            A valid ticket may be used in subsequent commands to provide proof to the TPM that the TPM has validated the signature over the message using the key referenced by *keyHandle*.

If *keyHandle* references an asymmetric key, only the public portion of the key needs to be loaded. If *keyHandle* references a symmetric key, both the public and private portions need to be loaded.

NOTE 2            The sensitive area of the symmetric object is required to allow verification of the symmetric signature (the HMAC).



22.1.2 Command and Response

**Table 93 — TPM2\_VerifySignature Command**

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_VerifySignature
TPMI_DH_OBJECT	keyHandle	handle of public key that will be used in the validation Auth Index: None
TPM2B_DIGEST	digest	digest of the signed message
TPMT_SIGNATURE	signature	signature to be tested

**Table 94 — TPM2\_VerifySignature Response**

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPMT_TK_VERIFIED	validation	

## 22.1.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "VerifySignature_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	<i>keyHandle</i> does not reference a signing key
TPM_RC_SIGNATURE	signature is not genuine
TPM_RC_SCHEME	<i>CryptVerifySignature ()</i>
TPM_RC_HANDLE	the input handle is not a sign key with private portion loaded

```

3  TPM_RC
4  TPM2_VerifySignature(
5      VerifySignature_In      *in,           // IN: input parameter list
6      VerifySignature_Out     *out          // OUT: output parameter list
7  )
8  {
9      TPM_RC                    result;
10     TPM2B_NAME                 name;
11     OBJECT                     *signObject;
12     TPMT_RH_HIERARCHY         hierarchy;
13
14     // Input Validation
15
16     // Get sign object pointer
17     signObject = ObjectGet(in->keyHandle);
18
19     // The object to validate the signature must be a signing key.
20     if(signObject->publicArea.objectAttributes.sign != SET)
21         return TPM_RC_ATTRIBUTES + RC_VerifySignature_keyHandle;
22
23     // If it doesn't have a sensitive area loaded
24     // then it can't be a keyed hash signing key
25     if( signObject->attributes.publicOnly == SET
26         && signObject->publicArea.type == TPM_ALG_KEYEDHASH
27         )
28         return TPM_RC_HANDLE + RC_VerifySignature_keyHandle;
29
30     // Validate Signature. A TPM_RC_BINDING, TPM_RC_SCHEME or TPM_RC_SIGNATURE
31     // error may be returned by CryptCVerifySignatrue()
32     result = CryptVerifySignature(in->keyHandle, &in->digest, &in->signature);
33     if(result != TPM_RC_SUCCESS)
34         return RcSafeAddToResult(result, RC_VerifySignature_signature);
35
36     // Command Output
37
38     hierarchy = ObjectGetHierarchy(in->keyHandle);
39     if( hierarchy == TPM_RH_NULL
40         || signObject->publicArea.nameAlg == TPM_ALG_NULL)
41     {
42         // produce empty ticket if hierarchy is TPM_RH_NULL or nameAlg is
43         // TPM_ALG_NULL
44         out->validation.tag = TPM_ST_VERIFIED;
45         out->validation.hierarchy = TPM_RH_NULL;
46         out->validation.digest.t.size = 0;
47     }
48     else
49     {

```

```
50     // Get object name that verifies the signature
51     name.t.size = ObjectGetName(in->keyHandle, name.t.name);
52     // Compute ticket
53     TicketComputeVerified(hierarchy, &in->digest, &name, &out->validation);
54 }
55
56 return TPM_RC_SUCCESS;
57 }
```

## 22.2 TPM2\_Sign

### 22.2.1 General Description

This command causes the TPM to sign an externally provided hash with the specified asymmetric signing key.

NOTE 1            Symmetric “signing” is done with an HMAC.

If *keyHandle* references a restricted signing key, then *validation* shall be provided indicating that the TPM performed the hash of the data and *validation* shall indicate that hashed data did not start with TPM\_GENERATED\_VALUE.

NOTE 2            If the hashed data did start with TPM\_GENERATED\_VALUE, then the validation will be a NULL ticket.

If the scheme of *keyHandle* is not TPM\_ALG\_NULL, then *inScheme* shall either be the same scheme as *keyHandle* or TPM\_ALG\_NULL.

If the scheme of *keyHandle* is TPM\_ALG\_NULL, the TPM will sign using *inScheme*; otherwise, it will sign using the scheme of *keyHandle*.

NOTE 3            When the signing scheme requires a hash algorithm, the hash is defined in the qualifying data of the scheme.

If *inScheme* is not a valid signing scheme for the type of *keyHandle* (or TPM\_ALG\_NULL), then the TPM shall return TPM\_RC\_SCHEME.

If the scheme of *keyHandle* is an anonymous *scheme*, then *inScheme* shall have the same scheme algorithm as *keyHandle* and *inScheme* will contain a counter value that will be used in the signing process.

As long as it is no larger than allowed, the *digest* parameter is not required to have any specific size but the signature operation may fail if *digest* is too large for the selected scheme.

If the *validation* parameter is not the Empty Buffer, then it will be checked even if the key referenced by *keyHandle* is not a restricted signing key.

22.2.2 Command and Response

Table 95 — TPM2\_Sign Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Sign
TPMI_DH_OBJECT	@keyHandle	Handle of key that will perform signing Auth Index: 1 Auth Role: USER
TPM2B_DIGEST	digest	digest to be signed
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the <i>scheme</i> for <i>keyHandle</i> is TPM_ALG_NULL
TPMT_TK_HASHCHECK	validation	proof that digest was created by the TPM If <i>keyHandle</i> is not a restricted signing key, then this may be a NULL Ticket with <i>tag</i> = TPM_ST_CHECKHASH.

Table 96 — TPM2\_Sign Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPMT_SIGNATURE	signature	the signature

## 22.2.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Sign_fp.h"
3  #include "Attest_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	key referenced by <i>keyHandle</i> is not a signing key
TPM_RC_BINDING	The public and private portions of the key are not properly bound.
TPM_RC_KEY	the key referenced by <i>keyHandle</i> is not a signing key
TPM_RC_SCHEME	<i>inScheme</i> is not compatible with <i>keyHandle</i> ; both <i>inScheme</i> and key's default scheme are empty; or <i>inScheme</i> is empty while key's default scheme requires explicit input scheme (split signing); or non-empty default key scheme differs from <i>inScheme</i>
TPM_RC_TICKET	<i>validation</i> is not a valid ticket
TPM_RC_VALUE	the value to sign is larger than allowed for the type of <i>keyHandle</i>

```

4  TPM_RC
5  TPM2_Sign(
6      Sign_In          *in,          // IN: input parameter list
7      Sign_Out         *out         // OUT: output parameter list
8  )
9  {
10     TPM_RC          result;
11     TPMT_TK_HASHCHECK ticket;
12     OBJECT          *signKey;
13
14     // Input Validation
15     // Get sign key pointer
16     signKey = ObjectGet(in->keyHandle);
17
18     // If validation is provided, or the key is restricted, check the ticket
19     if( in->validation.digest.t.size != 0
20        || signKey->publicArea.objectAttributes.restricted == SET)
21     {
22         // Compute and compare ticket
23         TicketComputeHashCheck(in->validation.hierarchy, &in->digest, &ticket);
24
25         if(!Memory2BEqual(&in->validation.digest.b, &ticket.digest.b))
26             return TPM_RC_TICKET + RC_Sign_validation;
27     }
28
29     // Command Output
30
31     // pick a scheme for sign. If the input sign scheme is not compatible with
32     // the default scheme, return an error.
33     result = CryptSelectSignScheme(in->keyHandle, &in->inScheme);
34     if(result != TPM_RC_SUCCESS)
35     {
36         if(result == TPM_RC_KEY)
37             return TPM_RC_KEY + RC_Sign_keyHandle;
38         else
39             return RcSafeAddToResult(result, RC_Sign_inScheme);
40     }
41
42     // Sign the hash. A TPM_RC_VALUE, TPM_RC_SCHEME, or TPM_RC_ATTRIBUTES
43     // error may be returned at this point

```

```
44     result = CryptSign(in->keyHandle, &in->inScheme, &in->digest, &out->signature);
45
46     return result;
47 }
```

## 23 Command Audit

### 23.1 Introduction

If a command has been selected for command audit, the command audit status will be updated when that command completes successfully. The digest is updated as:

$$commandAuditDigest_{new} := H_{auditAlg}(commandAuditDigest_{old} || cpHash || rpHash) \quad (5)$$

where

$H_{auditAlg}$	hash function using the algorithm of the audit sequence
$commandAuditDigest$	accumulated digest
$cpHash$	the command parameter hash
$rpHash$	the response parameter hash

TPM2\_Shutdown() cannot be audited but TPM2\_Startup() can be audited. If the  $cpHash$  of the TPM2\_Startup() is TPM\_SU\_STATE, that would indicate that a TPM2\_Shutdown() had been successfully executed.

TPM2\_SetCommandCodeAuditStatus() is always audited.

If the TPM is in Failure mode, command audit is not functional.



## 23.2 TPM2\_SetCommandCodeAuditStatus

### 23.2.1 General Description

This command may be used by the Privacy Administrator or platform to change the audit status of a command or to set the hash algorithm used for the audit digest, but not both at the same time.

If the *auditAlg* parameter is a supported hash algorithm and not the same as the current algorithm, then the TPM will check both *setList* and *clearList* are empty (zero length). If so, then the algorithm is changed, and the audit digest is cleared. If *auditAlg* is TPM\_ALG\_NULL or the same as the current algorithm, then the algorithm and audit digest are unchanged and the *setList* and *clearList* will be processed.

NOTE 1            Because the audit digest is cleared, the audit counter will increment the next time that an audited command is executed.

Use of TPM2\_SetCommandCodeAuditStatus() to change the list of audited commands is a audited event. If TPM\_CC\_SetCommandCodeAuditStatus is in *clearList*, it is ignored.

NOTE 2            Use of this command to change the audit hash algorithm is not audited and the digest is reset when the command completes. The change in the audit hash algorithm is the evidence that this command was used to change the algorithm.

The commands in *setList* indicate the commands that to be added to the list of audited commands and the commands in *clearList* indicate the commands that will no longer be audited. It is not an error if a command in *setList* is already audited or is not implemented. It is not an error if a command in *clearList* is not currently being audited or is not implemented.

If a command code is in both *setList* and *clearList*, then it will not be audited (that is, *setList* shall be processed first).

## 23.2.2 Command and Response

Table 97 — TPM2\_SetCommandCodeAuditStatus Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_SetCommandCodeAuditStatus {NV}
TPMI_RH_PROVISION	@auth	TPM_RH_ENDORSEMENT or TPM_RH_PLATFORM+{PP} Auth Index: 1 Auth Role: USER
TPMI_ALG_HASH+	auditAlg	hash algorithm for the audit digest; if TPM_ALG_NULL, then the hash is not changed
TPML_CC	setList	list of commands that will be added to those that will be audited
TPML_CC	clearList	list of commands that will no longer be audited

Table 98 — TPM2\_SetCommandCodeAuditStatus Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 23.2.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "SetCommandCodeAuditStatus_fp.h"
3  TPM_RC
4  TPM2_SetCommandCodeAuditStatus(
5      SetCommandCodeAuditStatus_In  *in          // IN: input parameter list
6  )
7  {
8      TPM_RC      result;
9      UINT32      i;
10     BOOL        changed = FALSE;
11
12
13     // The command needs NV update. Check if NV is available.
14     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
15     // this point
16     result = NvIsAvailable();
17     if(result != TPM_RC_SUCCESS)
18         return result;
19
20     // Internal Data Update
21
22     // Update hash algorithm
23     if( in->auditAlg != TPM_ALG_NULL
24         && in->auditAlg != gp.auditHashAlg)
25     {
26         // Can't change the algorithm and command list at the same time
27         if(in->setList.count != 0 || in->clearList.count != 0)
28             return TPM_RC_VALUE + RC_SetCommandCodeAuditStatus_auditAlg;
29
30         // Change the hash algorithm for audit
31         gp.auditHashAlg = in->auditAlg;
32
33         // Set the digest size to a unique value that indicates that the digest
34         // algorithm has been changed. The size will be cleared to zero in the
35         // command audit processing on exit.
36         gr.commandAuditDigest.t.size = 1;
37
38         // Save the change of command audit data (this sets g_updateNV so that NV
39         // will be updagted on exit.)
40         NvWriteReserved(NV_AUDIT_HASH_ALG, &gp.auditHashAlg);
41
42     } else {
43
44         // Process set list
45         for(i = 0; i < in->setList.count; i++)
46
47             // If change is made in CommandAuditSet, set changed flag
48             if(CommandAuditSet(in->setList.commandCodes[i]))
49                 changed = TRUE;
50
51         // Process clear list
52         for(i = 0; i < in->clearList.count; i++)
53             // If change is made in CommandAuditClear, set changed flag
54             if(CommandAuditClear(in->clearList.commandCodes[i]))
55                 changed = TRUE;
56
57         // if change was made to command list, update NV
58         if(changed)
59             // this sets g_updateNV so that NV will be updagted on exit.
60             NvWriteReserved(NV_AUDIT_COMMANDS, &gp.auditComands);

```

```
61     }  
62  
63     return TPM_RC_SUCCESS;  
64 }
```

## 24 Integrity Collection (PCR)

### 24.1 Introduction

In TPM 1.2, an Event was hashed using SHA-1 and then the 20-octet digest was extended to a PCR using TPM\_Extend(). This specification allows the use of multiple PCR at a given Index, each using a different hash algorithm. Rather than require that the external software generate multiple hashes of the Event with each being extended to a different PCR, the Event data may be sent to the TPM for hashing. This ensures that the resulting digests will properly reflect the algorithms chosen for the PCR even if the calling software is unable to implement the hash algorithm.

NOTE 1            There is continued support for software hashing of events with TPM2\_PCR\_Extend().

To support recording of an Event that is larger than the TPM input buffer, the caller may use the command sequence described in clause 1.

Change to a PCR requires authorization. The authorization may be with either an authorization value or an authorization policy. The platform-specific specifications determine which PCR may be controlled by policy. All other PCR are controlled by authorization.

If a PCR may be associated with a policy, then the algorithm ID of that policy determines whether the policy is to be applied. If the algorithm ID is not TPM\_ALG\_NULL, then the policy digest associated with the PCR must match the *policySession*→*policyDigest* in a policy session. If the algorithm ID is TPM\_ALG\_NULL, then no policy is present and the authorization requires an EmptyAuth.

If a platform-specific specification indicates that PCR are grouped, then all the PCR in the group use the same authorization policy or authorization value.

*PcrUpdateCounter* counter will be incremented on the successful completion of any command that modifies (Extends or resets) a PCR unless the platform-specific specification explicitly excludes the PCR from being counted.

NOTE 2            If a command causes PCR in multiple banks to change, the PCR Update Counter may be incremented either once or once for each bank.

A platform-specific specification may designate a set of PCR that are under control of the TCB. These PCR may not be modified without the proper authorization. Updates of these PCR shall not cause the PCR Update Counter to increment.

EXAMPLE            Updates of the TCB PCR will not cause the PCR update counter to increment because these PCR are changed at the whim of the TCB and are not intended to represent the trust state of the platform.

## 24.2 TPM2\_PCR\_Extend

### 24.2.1 General Description

This command is used to cause an update to the indicated PCR. The *digests* parameter contains one or more tagged digest value identified by an algorithm ID. For each digest, the PCR associated with *pcrHandle* is Extended into the bank identified by the tag (*hashAlg*).

EXAMPLE        A SHA1 digest would be Extended into the SHA1 bank and a SHA256 digest would be Extended into a SHA256 bank.

For each list entry, the TPM will check to see if *pcrNum* is implemented for that algorithm. If so, the TPM shall perform the following operation:

$$PCR.digest_{new}[pcrNum][alg] := H_{alg}(PCR.digest_{old}[pcrNum][alg] || data[alg].buffer) \quad (6)$$

where

$H_{alg}()$	hash function using the hash algorithm associated with the PCR instance
<i>PCR.digest</i>	the digest value in a PCR
<i>pcrNum</i>	the PCR numeric selector (equal to <i>pcrHandle</i> – TPM_RH_PCR0)
<i>alg</i>	the PCR algorithm selector for the digest
<i>data[alg].buffer</i>	the bank-specific data to be extended

If no digest value is specified for a bank, then the PCR in that bank are not modified.

NOTE 1        This allows consistent operation of the digests list for all of the Event recording commands.

If a digest is present and the PCR in that bank is not implemented, the digest value is not used.

NOTE 2        If the caller includes digests for algorithms that are not implemented, then the TPM will fail the call because the unmarshalling of *digests* will fail. Each of the entries in the list is a TPMT\_HA which is a hash algorithm followed by a digest. If the algorithm is not implemented, unmarshalling of the *hashAlg* will fail and the TPM will return TPM\_RC\_HASH.

If the TPM unmarshals the *hashAlg* of a list entry and the unmarshaled value is not a hash algorithm implemented on the TPM, the TPM shall return TPM\_RC\_HASH.

The *pcrHandle* parameter is allowed to reference TPM\_RH\_NULL. If so, the input parameters are processed but no action is taken by the TPM.

NOTE 3        This command allows a list of digests so that PCR in all banks may be updated in a single command. While the semantics of this command allow multiple extends to a single PCR bank, this is not the preferred use and the limit on the number of entries in the list make this use somewhat impractical.

24.2.2 Command and Response

**Table 99 — TPM2\_PCR\_Extend Command**

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PCR_Extend {NV}
TPMI_DH_PCR+	@pcrHandle	handle of the PCR Auth Handle: 1 Auth Role: USER
TPML_DIGEST_VALUES	digests	list of tagged digest values to be extended

**Table 100 — TPM2\_PCR\_Extend Response**

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	.

## 24.2.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PCR_Extend_fp.h"

```

Error Returns	Meaning
TPM_RC_LOCALITY	current command locality is not allowed to extend the PCR referenced by <i>pcrHandle</i>

```

3  TPM_RC
4  TPM2_PCR_Extend(
5      PCR_Extend_In      *in                // IN: input parameter list
6  )
7  {
8      TPM_RC              result;
9      UINT32              i;
10
11     // Input Validation
12
13     // NOTE: This function assumes that the unmarshaling function for 'digests' will
14     // have validated that all of the indicated hash algorithms are valid. If the
15     // hash algorithms are correct, the unmarshaling code will unmarshal a digest
16     // of the size indicated by the hash algorithm. If the overall size is not
17     // consistent, the unmarshaling code will run out of input data or have input
18     // data left over. In either case, it will cause an unmarshaling error and this
19     // function will not be called.
20
21     // For NULL handle, do nothing and return success
22     if(in->pcrHandle == TPM_RH_NULL)
23         return TPM_RC_SUCCESS;
24
25     // Check if the extend operation is allowed by the current command locality
26     if(!PCRIsExtendAllowed(in->pcrHandle))
27         return TPM_RC_LOCALITY;
28
29     // If PCR is state saved and we need to update orderlyState, check NV
30     // availability
31     if(PCRIsStateSaved(in->pcrHandle) && gp.orderlyState != SHUTDOWN_NONE)
32     {
33         result = NvIsAvailable();
34         if(result != TPM_RC_SUCCESS) return result;
35         g_clearOrderly = TRUE;
36     }
37
38     // Internal Data Update
39
40     // Iterate input digest list to extend
41     for(i = 0; i < in->digests.count; i++)
42     {
43         PCRExtend(in->pcrHandle, in->digests.digests[i].hashAlg,
44                 CryptGetHashDigestSize(in->digests.digests[i].hashAlg),
45                 (BYTE *) &in->digests.digests[i].digest);
46     }
47
48     return TPM_RC_SUCCESS;
49 }

```



## 24.3 TPM2\_PCR\_Event

### 24.3.1 General Description

This command is used to cause an update to the indicated PCR.

The data in *eventData* is hashed using the hash algorithm associated with each bank in which the indicated PCR has been allocated. After the data is hashed, the *digests* list is returned. If the *pcrHandle* references an implemented PCR and not TPM\_ALG\_NULL, *digests* list is processed as in TPM2\_PCR\_Extend().

A TPM shall support an *Event.size* of zero through 1,024 inclusive (*Event.size* is an octet count). An *Event.size* of zero indicates that there is no data but the indicated operations will still occur,

EXAMPLE 1 If the command implements PCR[2] in a SHA1 bank and a SHA256 bank, then an extend to PCR[2] will cause *eventData* to be hashed twice, once with SHA1 and once with SHA256. The SHA1 hash of *eventData* will be Extended to PCR[2] in the SHA1 bank and the SHA256 hash of *eventData* will be Extended to PCR[2] of the SHA256 bank.

On successful command completion, *digests* will contain the list of tagged digests of *eventData* that was computed in preparation for extending the data into the PCR. At the option of the TPM, the list may contain a digest for each bank, or it may only contain a digest for each bank in which *pcrHandle* is extant.

EXAMPLE 2 Assume a TPM that implements a SHA1 bank and a SHA256 bank and that PCR[22] is only implemented in the SHA1 bank. If *pcrHandle* references PCR[22], then *digests* may contain either a SHA1 and a SHA256 digest or just a SHA1 digest.

## 24.3.2 Command and Response

Table 101 — TPM2\_PCR\_Event Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PCR_Event {NV}
TPMI_DH_PCR+	@pcrHandle	Handle of the PCR Auth Handle: 1 Auth Role: USER
TPM2B_EVENT	eventData	Event data in sized buffer

Table 102 — TPM2\_PCR\_Event Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	.
TPML_DIGEST_VALUES	digests	

## 24.3.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PCR_Event_fp.h"

```

Error Returns	Meaning
TPM_RC_LOCALITY	current command locality is not allowed to extend the PCR referenced by <i>pcrHandle</i>

```

3  TPM_RC
4  TPM2_PCR_Event(
5      PCR_Event_In    *in,                // IN: input parameter list
6      PCR_Event_Out   *out               // OUT: output parameter list
7  )
8  {
9      TPM_RC          result;
10     HASH_STATE      hashState;
11     UINT32          i;
12     UINT16          size;
13
14     // Input Validation
15
16     // If a PCR extend is required
17     if(in->pcrHandle != TPM_RH_NULL)
18     {
19         // If the PCR is not allow to extend, return error
20         if(!PCRIsExtendAllowed(in->pcrHandle))
21             return TPM_RC_LOCALITY;
22
23         // If PCR is state saved and we need to update orderlyState, check NV
24         // availability
25         if(PCRIsStateSaved(in->pcrHandle) && gp.orderlyState != SHUTDOWN_NONE)
26         {
27             result = NvIsAvailable();
28             if(result != TPM_RC_SUCCESS) return result;
29             g_clearOrderly = TRUE;
30         }
31     }
32
33     // Internal Data Update
34
35     out->digests.count = HASH_COUNT;
36
37     // Iterate supported PCR bank algorithms to extend
38     for(i = 0; i < HASH_COUNT; i++)
39     {
40         TPM_ALG_ID hash = CryptGetHashAlgByIndex(i);
41         out->digests.digests[i].hashAlg = hash;
42         size = CryptStartHash(hash, &hashState);
43         CryptUpdateDigest2B(&hashState, &in->eventData.b);
44         CryptCompleteHash(&hashState, size,
45             (BYTE *) &out->digests.digests[i].digest);
46         if(in->pcrHandle != TPM_RH_NULL)
47             PCRExtend(in->pcrHandle, hash, size,
48                 (BYTE *) &out->digests.digests[i].digest);
49     }
50
51     return TPM_RC_SUCCESS;
52 }

```

## 24.4 TPM2\_PCR\_Read

### 24.4.1 General Description

This command returns the values of all PCR specified in *pcrSelect*.

The TPM will process the list of TPMS\_PCR\_SELECTION in *pcrSelectionIn* in order. Within each TPMS\_PCR\_SELECTION, the TPM will process the bits in the *pcrSelect* array in ascending PCR order (see Part 2 for definition of the PCR order). If a bit is SET, and the indicated PCR is present, then the TPM will add the digest of the PCR to the list of values to be returned in *pcrValue*.

The TPM will continue processing bits until all have been processed or until *pcrValues* would be too large to fit into the output buffer if additional values were added.

The returned *pcrSelectionOut* will have a bit SET in its *pcrSelect* structures for each value present in *pcrValues*.

The current value of the PCR Update Counter is returned in *pcrUpdateCounter*.

The returned list may be empty if none of the selected PCR are implemented.

NOTE If no PCR are returned from a bank, the selector for the bank will be present in *pcrSelectionOut*.

No authorization is required to read a PCR and any implemented PCR may be read from any locality.

## 24.4.2 Command and Response

Table 103 — TPM2\_PCR\_Read Command

Type	Name	Description
TPML_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PCR_Read
TPML_PCR_SELECTION	pcrSelectionIn	The selection of PCR to read

Table 104 — TPM2\_PCR\_Read Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
UINT32	pcrUpdateCounter	the current value of the PCR update counter
TPML_PCR_SELECTION	pcrSelectionOut	the PCR in the returned list
TPML_DIGEST	pcrValues	the contents of the PCR indicated in <i>pcrSelect</i> as tagged digests

### 24.4.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "PCR_Read_fp.h"
3  TPM_RC
4  TPM2_PCR_Read(
5      PCR_Read_In      *in,           // IN: input parameter list
6      PCR_Read_Out     *out          // OUT: output parameter list
7  )
8  {
9  // Command Output
10
11     // Call PCR read function.  input pcrSelectionIn parameter could be changed
12     // to reflect the actual PCR being returned
13     PCRRead(&in->pcrSelectionIn, &out->pcrValues, &out->pcrUpdateCounter);
14
15     out->pcrSelectionOut = in->pcrSelectionIn;
16
17     return TPM_RC_SUCCESS;
18 }
```

## 24.5 TPM2\_PCR\_Allocate

### 24.5.1 General Description

This command is used to set the desired PCR allocation of PCR and algorithms. This command requires *platformAuth*.

The TPM will evaluate the request and, if sufficient memory is available for the requested allocation, the TPM will store the allocation request for use during the next TPM2\_Startup(TPM\_SU\_CLEAR) operation. The PCR allocation in place when this command is executed will be retained until the next TPM2\_Startup(TPM\_SU\_CLEAR).

If no allocation is specified for a bank, then no PCR will be allocated to that bank. If a bank is listed more than once, then the last selection in the *pcrAllocation* list is the one that the TPM will attempt to allocate.

This command shall not allocate more PCR in any bank than there are PCR attribute definitions. The PCR attribute definitions indicate how a PCR is to be managed – if it is resettable, the locality for update, etc. In the response to this command, the TPM returns the maximum number of PCR allowed for any bank.

If the command is properly authorized, it will return SUCCESS even though the request fails. This is to allow the TPM to return information about the size needed for the requested allocation and the size available. If the *sizeNeeded* parameter in the return is less than or equal to the *sizeAvailable* parameter, then the *allocationSuccess* parameter will be YES.

After this command, TPM2\_Shutdown() is only allowed to have a *startupType* equal to TPM\_SU\_CLEAR.

NOTE Even if this command does not cause the PCR allocation to change, the TPM cannot have its state saved. This is done in order to simplify the implementation. There is no need to optimize this command as it is not expected to be used more than once in the lifetime of the TPM (it can be used any number of times but there is no justification for optimization).

## 24.5.2 Command and Response

Table 105 — TPM2\_PCR\_Allocate Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PCR_Allocate {NV}
TPMI_RH_PLATFORM	@authHandle	TPM_RH_PLATFORM+{PP} Auth Index: 1 Auth Role: USER
TPML_PCR_SELECTION	pcrAllocation	the requested allocation

Table 106 — TPM2\_PCR\_Allocate Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPMI_YES_NO	allocationSuccess	YES if the allocation succeeded
UINT32	maxPCR	maximum number of PCR that may be in a bank
UINT32	sizeNeeded	number of octets required to satisfy the request
UINT32	sizeAvailable	Number of octets available. Computed before the allocation.



## 24.5.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "PCR_Allocate_fp.h"
3  TPM_RC
4  TPM2_PCR_Allocate(
5      PCR_Allocate_In    *in,           // IN: input parameter list
6      PCR_Allocate_Out   *out          // OUT: output parameter list
7  )
8  {
9      TPM_RC    result;
10
11     // The command needs NV update. Check if NV is available.
12     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
13     // this point.
14     // Note: These codes are not listed in the return values above because it is
15     // an implementation choice to check in this routine rather than in a common
16     // function that is called before these actions are called. These return values
17     // are described in the Response Code section of Part 3.
18     result = NvIsAvailable();
19     if(result != TPM_RC_SUCCESS)
20         return result;
21
22     // Command Output
23
24     // Call PCR Allocation function.
25     out->allocationSuccess = PCRAAllocate(&in->pcrAllocation, &out->maxPCR,
26                                         &out->sizeNeeded, &out->sizeAvailable);
27
28     // if re-configuration succeeds, set the flag to indicate PCR configuration is
29     // going to be changed in next boot
30     if(out->allocationSuccess == YES)
31         g_pcrReConfig = TRUE;
32
33     return TPM_RC_SUCCESS;
34 }
```

## 24.6 TPM2\_PCR\_SetAuthPolicy

### 24.6.1 General Description

This command is used to associate a policy with a PCR or group of PCR. The policy determines the conditions under which a PCR may be extended or reset.

A policy may only be associated with a PCR that has been defined by a platform-specific specification as allowing a policy. If the TPM implementation does not allow a policy for *pcrNum*, the TPM shall return TPM\_RC\_VALUE.

A platform-specific specification may group PCR so that they share a common policy. In such case, a *pcrNum* that selects any of the PCR in the group will change the policy for all PCR in the group.

The policy setting is persistent and may only be changed by TPM2\_PCR\_SetAuthPolicy() or by TPM2\_ChangePPS().

Before this command is first executed on a TPM or after TPM2\_ChangePPS(), the access control on the PCR will be set to the default value defined in the platform-specific specification.

NOTE 1           It is expected that the typical default will be with the policy hash set to TPM\_ALG\_NULL and an Empty Buffer for the *authPolicy* value. This will allow an *EmptyAuth* to be used as the authorization value.

If the size of the data buffer in *authPolicy* is not the size of a digest produced by *hashAlg*, the TPM shall return TPM\_RC\_SIZE.

NOTE 2           If *hashAlg* is TPM\_ALG\_NULL, then the size is required to be zero.

This command requires platformAuth/platformPolicy.

NOTE 3           If the PCR is in multiple policy sets, the policy will be changed in only one set. The set that is changed will be implementation dependent.

24.6.2 Command and Response

**Table 107 — TPM2\_PCR\_SetAuthPolicy Command**

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PCR_SetAuthPolicy {NV}
TPMI_RH_PLATFORM	@authHandle	TPM_RH_PLATFORM+{PP} Auth Index: 1 Auth Role: USER
TPM2B_DIGEST	authPolicy	the desired <i>authPolicy</i>
TPMI_ALG_HASH+	policyDigest	the digest of the policy
TPMI_DH_PCR	pcrNum	the PCR for which the policy is to be set

**Table 108 — TPM2\_PCR\_SetAuthPolicy Response**

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 24.6.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PCR_SetAuthPolicy_fp.h"

```

Error Returns	Meaning
TPM_RC_SIZE	size of <i>authPolicy</i> is not the size of a digest produced by <i>policyDigest</i>
TPM_RC_VALUE	PCR referenced by <i>pcrNum</i> is not a member of a PCR policy group

```

3  TPM_RC
4  TPM2_PCR_SetAuthPolicy(
5      PCR_SetAuthPolicy_In    *in                // IN: input parameter list
6  )
7  {
8      UINT32    groupIndex;
9
10     TPM_RC    result;
11
12     // The command needs NV update. Check if NV is available.
13     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
14     // this point
15     result = NvIsAvailable();
16     if(result != TPM_RC_SUCCESS) return result;
17
18     // Input Validation:
19
20     // Check the authPolicy consistent with hash algorithm
21     if(in->authPolicy.t.size != CryptGetHashDigestSize(in->policyDigest))
22         return TPM_RC_SIZE + RC_PCR_SetAuthPolicy_authPolicy;
23
24     // If PCR does not belong to a policy group, return TPM_RC_VALUE
25     if(!PCRBelongsPolicyGroup(in->pcrNum, &groupIndex))
26         return TPM_RC_VALUE + RC_PCR_SetAuthPolicy_pcrNum;
27
28     // Internal Data Update
29
30     // Set PCR policy
31     gp.pcrPolicies.hashAlg[groupIndex] = in->policyDigest;
32     gp.pcrPolicies.policy[groupIndex] = in->authPolicy;
33
34     // Save new policy to NV
35     NvWriteReserved(NV_PCR_POLICIES, &gp.pcrPolicies);
36
37     return TPM_RC_SUCCESS;
38 }

```

## 24.7 TPM2\_PCR\_SetAuthValue

### 24.7.1 General Description

This command changes the *authValue* of a PCR or group of PCR.

An *authValue* may only be associated with a PCR that has been defined by a platform-specific specification as allowing an authorization value. If the TPM implementation does not allow an authorization for *pcrNum*, the TPM shall return TPM\_RC\_VALUE. A platform-specific specification may group PCR so that they share a common authorization value. In such case, a *pcrNum* that selects any of the PCR in the group will change the *authValue* value for all PCR in the group.

The authorization setting is set to EmptyAuth on each STARTUP(CLEAR) or by TPM2\_Clear(). The authorization setting is preserved by SHUTDOWN(STATE).

## 24.7.2 Command and Response

Table 109 — TPM2\_PCR\_SetAuthValue Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PCR_SetAuthValue
TPMI_DH_PCR	@pcrHandle	handle for a PCR that may have an authorization value set Auth Index: 1 Auth Role: USER
TPM2B_DIGEST	auth	the desired authorization value

Table 110 — TPM2\_PCR\_SetAuthValue Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 24.7.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PCR_SetAuthValue_fp.h"

```

Error Returns	Meaning
TPM_RC_VALUE	PCR referenced by <i>pcrHandle</i> is not a member of a PCR authorization group

```

3  TPM_RC
4  TPM2_PCR_SetAuthValue(
5      PCR_SetAuthValue_In    *in           // IN: input parameter list
6  )
7  {
8      UINT32    groupIndex;
9      TPM_RC    result;
10
11 // Input Validation:
12
13 // If PCR does not belong to an auth group, return TPM_RC_VALUE
14 if(!PCRBelongsAuthGroup(in->pcrHandle, &groupIndex))
15     return TPM_RC_VALUE;
16
17 // The command may cause the orderlyState to be cleared due to the update of
18 // state clear data.  If this is the case, Check if NV is available.
19 // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
20 // this point
21 if(gp.orderlyState != SHUTDOWN_NONE)
22 {
23     result = NvIsAvailable();
24     if(result != TPM_RC_SUCCESS) return result;
25     g_clearOrderly = TRUE;
26 }
27
28 // Internal Data Update
29
30 // Set PCR authValue
31 gc.pcrAuthValues.auth[groupIndex] = in->auth;
32
33 return TPM_RC_SUCCESS;
34 }

```

## 24.8 TPM2\_PCR\_Reset

### 24.8.1 General Description

If the attribute of a PCR allows the PCR to be reset and proper authorization is provided, then this command may be used to set the PCR to zero. The attributes of the PCR may restrict the locality that can perform the reset operation.

NOTE 1            The definition of TPMI\_DH\_PCR in Part 2 indicates that if *pcrHandle* is out of the allowed range for PCR, then the appropriate return value is TPM\_RC\_VALUE.

If *pcrHandle* references a PCR that cannot be reset, the TPM shall return TPM\_RC\_LOCALITY.

NOTE 2            TPM\_RC\_LOCALITY is returned because the reset attributes are defined on a per-locality basis.



24.8.2 Command and Response

Table 111 — TPM2\_PCR\_Reset Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PCR_Reset {NV}
TPMI_DH_PCR	@pcrHandle	the PCR to reset Auth Index: 1 Auth Role: USER

Table 112 — TPM2\_PCR\_Reset Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 24.8.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PCR_Reset_fp.h"

```

Error Returns	Meaning
TPM_RC_LOCALITY	current command locality is not allowed to reset the PCR referenced by <i>pcrHandle</i>

```

3  TPM_RC
4  TPM2_PCR_Reset(
5      PCR_Reset_In    *in                // IN: input parameter list
6  )
7  {
8      TPM_RC    result;
9
10     // Input Validation
11
12     // Check if the reset operation is allowed by the current command locality
13     if(!PCRIsResetAllowed(in->pcrHandle))
14         return TPM_RC_LOCALITY;
15
16     // If PCR is state saved and we need to update orderlyState, check NV
17     // availability
18     if(PCRIsStateSaved(in->pcrHandle) && gp.orderlyState != SHUTDOWN_NONE)
19     {
20         result = NvIsAvailable();
21         if(result != TPM_RC_SUCCESS)
22             return result;
23         g_clearOrderly = TRUE;
24     }
25
26     // Internal Data Update
27
28     // Reset selected PCR in all banks to 0
29     PCRSetValue(in->pcrHandle, 0);
30
31     // Indicate that the PCR changed so that pcrCounter will be incremented if
32     // necessary.
33     PCRChanged(in->pcrHandle);
34
35     return TPM_RC_SUCCESS;
36 }

```

## 24.9 \_TPM\_Hash\_Start

### 24.9.1 Description

This indication from the TPM interface indicates the start of a dynamic Core Root of Trust for Measurement (D-CRTM) measurement sequence. On receipt of this indication, the TPM will initialize an Event sequence context.

If no object memory is available for creation of the sequence context, the TPM will flush the context of an object so that creation of the Event sequence context will always succeed.

A platform-specific specification may allow this indication before TPM2\_Startup().

**NOTE** If this indication occurs after TPM2\_Startup(), it is the responsibility of software to ensure that an object context slot is available or to deal with the consequences of having the TPM select an arbitrary object to be flushed. If this indication occurs before TPM2\_Startup() then all context slots are available.

## 24.9.2 Detailed Actions

```
1 #include "InternalRoutines.h"
```

This function is called to process a `_TPM_Hash_Start()` indication.

```
2 void
3 _TPM_Hash_Start(void)
4 {
5     TPM_RC          result;
6     TPMI_DH_OBJECT handle;
7
8     // If a DRTM sequence object exists, terminate it.
9     if(g_DRTMHandle != TPM_RH_UNASSIGNED)
10        ObjectTerminateEvent();
11
12    // Create an event sequence object and store the handle in global
13    // g_DRTMHandle. A TPM_RC_OBJECT_MEMORY error may be returned at this point
14    // The null value for the 'auth' parameter will cause the sequence structure to
15    // be allocated without being set as present. This keeps the sequence from
16    // being left behind if the sequence is terminated early.
17    result = ObjectCreateEventSequence(NULL, &g_DRTMHandle);
18
19    // If a free slot was not available, then free up a slot.
20    if(result != TPM_RC_SUCCESS)
21    {
22        // An implementation does not need to have a fixed relationship between
23        // slot numbers and handle numbers. To handle the general case, scan for
24        // a handle that is assigned an free it for the DRTM sequence.
25        // In the reference implementation, the relationship between handles and
26        // slots is fixed. So, if the call to ObjectCreateEvenSequence()
27        // failed indicating that all slots are occupied, then the first handle we
28        // are going to check (TRANSIENT_FIRST) will be occupied. It will be freed
29        // so that it can be assigned for use as the DRTM sequence object.
30        for(handle = TRANSIENT_FIRST; handle < TRANSIENT_LAST; handle++)
31        {
32            // try to flush the first object
33            if(ObjectIsPresent(handle))
34            {
35                ObjectFlush(handle);
36                break;
37            }
38        }
39
40        // Try to create an event sequence object again. This time, we must
41        // succeed.
42        result = ObjectCreateEventSequence(NULL, &g_DRTMHandle);
43        pAssert(result == TPM_RC_SUCCESS);
44    }
45
46    return;
47 }
```

## 24.10 \_TPM\_Hash\_Data

### 24.10.1 Description

This indication from the TPM interface indicates arrival of one or more octets of data that are to be included in the Core Root of Trust for Measurement (CRTM) sequence context created by the \_TPM\_Hash\_Start indication. The context holds data for each hash algorithm for each PCR bank implemented on the TPM.

If no DRTM Event Sequence context exists, this indication is discarded and no other action is performed.

## 24.10.2 Detailed Actions

```
1 #include "InternalRoutines.h"
2 #include "Platform.h"
```

This function is called to process a `_TPM_Hash_Data()` indication.

```
3 void
4 _TPM_Hash_Data(
5     UINT32      dataSize,          // IN: size of data to be extend
6     BYTE        *data             // IN: data buffer
7 )
8 {
9     UINT32      i;
10    HASH_OBJECT *hashObject;
11
12    // If there is no DRTM sequence object, then _TPM_Hash_Start
13    // was not called so this function returns without doing
14    // anything.
15    if(g_DRTMHandle == TPM_RH_UNASSIGNED)
16        return;
17
18    hashObject = (HASH_OBJECT *)ObjectGet(g_DRTMHandle);
19    pAssert(hashObject->attributes.eventSeq);
20
21    // For each of the implemented hash algorithms, update the digest with the
22    // data provided. NOTE: the implementation could be done such that the TPM
23    // only computes the hash for the banks that contain the DRTM PCR.
24    for(i = 0; i < HASH_COUNT; i++)
25    {
26        // Update sequence object
27        CryptUpdateDigest(&hashObject->state.hashState[i], dataSize, data);
28    }
29
30    return;
31 }
```

## 24.11 \_TPM\_Hash\_End

### 24.11.1 Description

This indication from the TPM interface indicates the end of the CRTM measurement. This indication is discarded and no other action performed if the TPM does not contain a CRTM Event sequence context.

NOTE A CRTM Event Sequence context is created by `_TPM_Hash_Start()`.

If the CRTM Event sequence occurs after `TPM2_Startup()`, the TPM will set all of the PCR designated in the platform-specific specifications as resettable by by this event to the value indicated in the platform specific specification, and increment *restartCount*. The TPM will then Extend the Event Sequence digest/digests into the designated, DRTM PCR.

$$\text{PCR}[\text{DRTM}][\text{hashAlg}] := \mathbf{H}_{\text{hashAlg}}(\text{initial\_value} || \mathbf{H}_{\text{hashAlg}}(\text{hash\_data})) \quad (7)$$

where

<i>DRTM</i>	index for CRTM PCR designated by a platform-specific specification
<i>hashAlg</i>	hash algorithm associated with a bank of PCR
<i>initial_value</i>	initialization value specified in the platform-specific specification (should be 0...0)
<i>hash_data</i>	all the octets of data received in <code>_TPM_Hash_Data</code> indications

A `_TPM_Hash_End` indication that occurs after `TPM2_Startup()` will increment *pcrUpdateCounter* unless a platform-specific specification excludes modifications of PCR[DRTM] from causing an increment.

A platform-specific specification may allow an H-CRTM Event Sequence before `TPM2_Startup()`. If so, `_TPM_Hash_End` will complete the digest, initialize PCR[0] with a digest-size value of 4, and then extend the H-CRTM Event Sequence data into PCR[0].

$$\text{PCR}[0][\text{hashAlg}] := \mathbf{H}_{\text{hashAlg}}(0\dots04 || \mathbf{H}_{\text{hashAlg}}(\text{hash\_data})) \quad (8)$$

NOTE The entire sequence of `_TPM_Hash_Start`, `_TPM_Hash_Data`, and `_TPM_Hash_End` are required to complete before `TPM2_Startup()` or the sequence will have no effect on the TPM.

## 24.11.2 Detailed Actions

```
1 #include "InternalRoutines.h"
```

This function is called to process a `_TPM_Hash_End()` indication.

```
2 void
3 _TPM_Hash_End(void)
4 {
5
6     UINT32          i;
7     TPM2B_DIGEST    digest;
8     HASH_OBJECT     *hashObject;
9     TPMI_DH_PCR     pcrHandle;
10
11     // If the DRTM handle is not being used, then either _TPM_Hash_Start has not
12     // been called, _TPM_Hash_End was previously called, or some other command
13     // was executed and the sequence was aborted.
14     if(g_DRTMHandle == TPM_RH_UNASSIGNED)
15         return;
16
17     // Get DRTM sequence object
18     hashObject = (HASH_OBJECT *)ObjectGet(g_DRTMHandle);
19
20
21     // Is this _TPM_Hash_End after Startup or before
22     if(TPMIsStarted())
23     {
24         // After
25
26         // Reset the DRTM PCR
27         PCRResetDynamics();
28
29         // Extend the DRTM PCR.
30         pcrHandle = PCR_FIRST + DRTM_PCR;
31
32         // DRTM sequence increments restartCount
33         gr.restartCount++;
34     }
35     else
36     {
37         pcrHandle = PCR_FIRST;
38
39         // This is pre-startup so set PCR[0] to 4
40         PCRSetValue(0 + PCR_FIRST, 4);
41     }
42
43
44     // Complete hash and extend PCR
45     for(i = 0; i < HASH_COUNT; i++)
46     {
47         TPMI_ALG_HASH    hash = CryptGetHashAlgByIndex(i);
48
49         // Complete hash
50         digest.t.size = CryptGetHashDigestSize(hash);
51         CryptCompleteHash2B(&hashObject->state.hashState[i], &digest.b);
52
53         // Extend PCR
54         PCRExtend(pcrHandle, hash, digest.t.size, digest.t.buffer);
55     }
56
57     // Flush sequence object.
```



```
58     ObjectFlush(g_DRTMHandle);
59
60     g_DRTMHandle = TPM_RH_UNASSIGNED;
61
62     return;
63 }
```

## 25 Enhanced Authorization (EA) Commands

### 25.1 Introduction

The commands in this clause 1 are used for policy evaluation. When successful, each command will update the *policySession*→*policyDigest* in a policy session context in order to establish that the authorizations required to use an object have been provided. Many of the commands will also modify other parts of a policy context so that the caller may constrain the scope of the authorization that is provided.

NOTE 1 Many of the terms used in this clause are described in detail in Part 1 and are not redefined in this clause.

The *policySession* parameter of the command is the handle of the policy session context to be modified by the command.

If the *policySession* parameter indicates a trial policy session, then the *policySession*→*policyDigest* will be updated and the indicated validations are not performed.

NOTE 2 A policy session is a trial policy by `TPM2_StartAuthSession(sessionType = TPM_SE_TRIAL)`.

NOTE 3 Unless there is an unmarshaling error in the parameters of the command, these commands will return `TPM_RC_SUCCESS` when *policySession* references a trial session.

NOTE 4 Policy context other than the *policySession*→*policyDigest* may be updated for a trial policy but it is not required.

## 25.2 Signed Authorization Actions

### 25.2.1 Introduction

The TPM2\_PolicySigned, TPM\_PolicySecret, and TPM2\_PolicyTicket commands use many of the same functions. This clause consolidates those functions to simplify the document and to ensure uniformity of the operations.

### 25.2.2 Policy Parameter Checks

These parameter checks will be performed when indicated in the description of each of the commands:

- a) *nonceTPM* – If this parameter is not the Empty Buffer, and it does not match *policySession→nonceTPM*, then the TPM shall return TPM\_RC\_VALUE.
- b) *expiration* – If this parameter is not zero, then it is compared to the time in seconds since the *policySession→nonceTPM* was generated. If more time has passed than indicated in *expiration*, the TPM shall return TPM\_RC\_EXPIRED.
- c) *timeout* – This parameter is compared to the current TPM time. If *policySession→timeout* is in the past, then the TPM shall return TPM\_RC\_EXPIRED.

NOTE 1            The *expiration* parameter is present in the TPM2\_PolicySigned and TPM2\_PolicySecret command and *timeout* is the analogous parameter in the TPM2\_PolicyTicket command.

- d) *cpHashA* – If this parameter is not an Empty Buffer

NOTE 2            *CpHashA* is the hash of the command to be executed using this policy session in the authorization. The algorithm used to compute this hash is required to be the algorithm of the policy session.

- 1) the TPM shall return TPM\_RC\_CPHASH if *policySession→cpHash* does not have its default value or the contents of *policySession→cpHash* are not the same as *cpHashA*; or

NOTE 3            *CpHash* is the expected *cpHash* value held in the policy session context.

- 2) the TPM shall return TPM\_RC\_SIZE if *cpHashA* is not the same size as *policySession→policyDigest*.

NOTE 4            *PolicySession→policyDigest* is the size of the digest produced by the hash algorithm used to compute *policyDigest*.

### 25.2.3 PolicyDigest Update Function (PolicyUpdate())

This is the update process for  $policySession \rightarrow policyDigest$  used by TPM2\_PolicySigned(), TPM2\_PolicySecret(), TPM2\_PolicyTicket(), and TPM2\_PolicyAuthorize(). The function prototype for the update function is:

$$\mathbf{PolicyUpdate}(commandCode, arg2, arg3) \quad (9)$$

where

$arg2$  a TPM2B\_NAME

$arg3$  a TPM2B

These parameters are used to update  $policySession \rightarrow policyDigest$  by

$$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || commandCode || arg2.name) \quad (10)$$

followed by

$$policyDigest_{new+1} := H_{policyAlg}(policyDigest_{new} || arg3.buffer) \quad (11)$$

where

$H_{policyAlg}()$  the hash algorithm chosen when the policy session was started

NOTE 1 If  $arg3$  is a TPM2B\_NAME, then  $arg3.buffer$  will actually be an  $arg3.name$ .

NOTE 2 The  $arg2.size$  and  $arg3.size$  fields are not included in the hashes.

NOTE 3 **PolicyUpdate()** uses two hashes because  $arg2$  and  $arg3$  are variable-sized and the concatenation of  $arg2$  and  $arg3$  in a single hash could produce the same digest even though  $arg2$  and  $arg3$  are different. Processing of the arguments separately in different Extend operation insures that the digest produced by **PolicyUpdate()** will be different if  $arg2$  and  $arg3$  are different.

### 25.2.4 Policy Context Updates

When a policy command modifies some part of the policy session context other than the *policySession*→*policyDigest*, the following rules apply.

- ***cpHash*** – this parameter may only be changed if it contains its initialization value (an Empty String). If *cpHash* is not the Empty String when a policy command attempts to update it, the TPM will return an error (TPM\_RC\_CPHASH) if the current and update values are not the same.
- ***timeOut*** – this parameter may only be changed to a smaller value. If a command attempts to update this value with a larger value (longer into the future), the TPM will discard the update value. This is not an error condition.
- ***commandCode*** – once set by a policy command, this value may not be change except by TPM2\_PolicyRestart(). If a policy command tries to change this to a different value, an error is returned (TPM\_RC\_POLICY\_CC).
- ***pcrUpdateCounter*** – this parameter is updated by TPM2\_PolicyPCR(). This value may only be set once during a policy. Each time TPM2\_PolicyPCR() executes, it checks to see if *policySession*→*pcrUpdateCounter* has its default state indicating that this is the first TPM2\_PolicyPCR(). If it has its default value, then *policySession*→*pcrUpdateCounter* is set to the current value of *pcrUpdateCounter*. If *policySession*→*pcrUpdateCounter* does not have its default value and its value is not the same as *pcrUpdateCounter*, the TPM shall return TPM\_RC\_PCR\_CHANGED.

NOTE                    If this parameter and *pcrUpdateCounter* are not the same, it indicates that PCR have changed since checked by the previous TPM2\_PolicyPCR(). Since they have changed, the previous PCR validation is no longer valid.

- ***commandLocality*** – this parameter is the logical AND of all enabled localities. All localities are enabled for a policy when the policy session is created. TPM2\_PolicyLocalities() selectively disables localities. Once use of a policy for a locality has been disabled, it cannot be enabled except by TPM2\_PolicyRestart().
- ***isPPRequired*** – once SET, this parameter may only be CLEARED by TPM2\_PolicyRestart().
- ***isAuthValueNeeded*** – once SET, this parameter may only be CLEARED by TPM2\_PolicyPassword() or TPM2\_PolicyRestart().
- ***isPasswordNeeded*** – once SET, this parameter may only be CLEARED by TPM2\_PolicyAuthValue() or TPM2\_PolicyRestart(),

NOTE                    Both TPM2\_PolicyAuthValue() and TPM2\_PolicyPassword() change *policySession*→*policyDigest* in the same way. The different commands simply indicate to the TPM the format used for the *authValue* (HMAC or clear text). Both commands could be in the same policy. The final instance of these commands determines the format.

### 25.2.5 Policy Ticket Creation

If for TPM2\_PolicySigned() or TPM2\_PolicySecret() the caller specified a non-zero value for *expiration*, and the *nonceTPM* is an Empty Buffer, then the TPM will return a ticket that includes a value to indicate when the authorization expires. The required computation for the digest in the authorization ticket is:

$$\mathbf{HMAC}(\mathit{proof}, \mathbf{H}_{\mathit{policyAlg}}(\mathit{ticketType} || \mathit{timeout} || \mathit{cpHashA} || \mathit{policyRef} || \mathit{authObject} \rightarrow \mathit{Name})) \quad (12)$$

where

*proof* secret associated with the storage primary seed (SPS) of the TPM

$\mathbf{H}_{\mathit{policyAlg}}$  hash function using the hash algorithm associated with the policy session

*ticketType* either TPM\_ST\_AUTH\_SECRET or TPM\_ST\_AUTH\_SIGNED, used to indicate type of the ticket

NOTE 1 If the ticket is produced by TPM2\_PolicySecret() then *ticketType* is TPM\_ST\_AUTH\_SECRET and if produced by TPM2\_PolicySigned() then *ticketType* is TPM\_ST\_AUTH\_SIGNED.

*timeout* implementation-specific representation of the expiration time of the ticket

NOTE 2 *Timeout* is not the same as *expiration*. The *expiration* value in the *aHash* is a relative time, using the creation time of the authorization session (TPM2\_StartAuthSession()) as its reference. The *timeout* parameter is an absolute time, using TPM *Clock* as the reference.

*cpHashA* the command parameter digest for the command being authorized; computed using the hash algorithm of the policy session

*policyRef* the commands that use this function have a *policyRef* parameter and the value of that parameter is used here

*authObject* → *Name* Name associated with the *authObject* parameter

## 25.3 TPM2\_PolicySigned

### 25.3.1 General Description

This command includes a signed authorization in a policy. The command ties the policy to a signing key by including the Name of the signing key in the *policyDigest*

If *policySession* is a trial session, the TPM will not check the signature and will update *policySession*→*policyDigest* as described in 25.2.3 as if a properly signed authorization was received; but no ticket will be produced.

If *policySession* is not a trial session, the TPM will validate *auth* and only perform the update if it is a valid signature over the fields of the command.

The authorizing object will sign a digest of the authorization qualifiers: *nonceTPM*, *expiration*, *cpHashA*, and *policyRef*. The digest is computed as:

$$aHash := H_{authAlg}(nonceTPM || expiration || cpHashA || policyRef) \quad (13)$$

where

$H_{authAlg}()$  the hash associated with the auth parameter of this command

NOTE 1 Each signature and key combination indicates the scheme and each scheme has an associated hash.

*nonceTPM* the nonceTPM parameter from the TPM2\_StartAuthSession() response. If the authorization is not limited to this session, the size of this value is zero.

*expiration* time limit on authorization set by authorizing object. This 32-bit value is set to zero if the expiration time is not being set.

*cpHashA* digest of the command parameters for the command being approved using the hash algorithm of the policy session. Set to an EmptyAuth if the authorization is not limited to a specific command.

NOTE 2 This is not the *cpHash* of this TPM2\_PolicySigned() command.

*policyRef* an opaque value determined by the authorizing entity. Set to the Empty Buffer if no value is present.

EXAMPLE The computation for an *aHash* if there are no restrictions is:

$$aHash := H_{authAlg}(00\ 00\ 00\ 00_{16})$$

which is the hash of an expiration time of zero.

The *aHash* is signed by the private key associated with key. The signature and signing parameters are combined to create the *auth* parameter.

The TPM will perform the parameter checks listed in 25.2.2

If the parameter checks succeed, the TPM will construct a test digest (*tHash*) over the provided parameters using the same formulation a shown in equation (13) above.

If *tHash* does not match the digest of the signed *aHash*, then the authorization fails and the TPM shall return TPM\_RC\_POLICY\_FAIL and make no change to *policySession*→*policyDigest*.

When all validations have succeeded, *policySession*→*policyDigest* is updated by **PolicyUpdate()** (see 25.2.3).

**PolicyUpdate**(TPM\_CC\_PolicySigned, *authObject*→*Name*, *policyRef*) (14)

If the *cpHashA* parameter is not an Empty Buffer, it is copied to *policySession*→*cpHash*.

The TPM will optionally produce a ticket as described in 25.2.5.

Authorization to use *authObject* is not required.



## 25.3.2 Command and Response

Table 113 — TPM2\_PolicySigned Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicySigned
TPMI_DH_OBJECT	authObject	handle for a public key that will validate the signature Auth Index: None
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None
TPM2B_NONCE	nonceTPM	the policy nonce for the session If the nonce is not included in the authorization qualification, this field is the Empty Buffer.
TPM2B_DIGEST	cpHashA	digest of the command parameters to which this authorization is limited This is not the <i>cpHash</i> for this command but the <i>cpHash</i> for the command to which this policy session will be applied. If it is not limited, the parameter will be the Empty Buffer.
TPM2B_NONCE	policyRef	a reference to a policy relating to the authorization – may be the Empty Buffer Size is limited to be no larger than the nonce size supported on the TPM.
UINT32	expiration	time when authorization will expire, measured in seconds from the time that <i>nonceTPM</i> was generated If <i>expiration</i> is zero, a NULL Ticket is returned.
TPMT_SIGNATURE	auth	signed authorization (not optional)

Table 114 — TPM2\_PolicySigned Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_TIMEOUT	timeout	implementation-specific time value, used to indicate to the TPM when the ticket expires NOTE If <i>policyTicket</i> is a NULL Ticket, then this shall be the Empty Buffer.
TPMT_TK_AUTH	policyTicket	produced if the command succeeds and <i>expiration</i> in the command was non-zero; this ticket will use the TPM2B_ST_AUTH_SIGNED structure tag

## 25.3.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Policy_spt_fp.h"
3  #include "PolicySigned_fp.h"

```

Error Returns	Meaning
TPM_RC_CPHASH	<i>cpHash</i> was previously set to a different value
TPM_RC_EXPIRED	<i>expiration</i> indicates a time in the past
TPM_RC_HANDLE	<i>authObject</i> need to have sensitive portion loaded
TPM_RC_KEY	<i>authObject</i> is not a signing scheme
TPM_RC_NONCE	<i>nonceTPM</i> is not the nonce associated with the <i>policySession</i>
TPM_RC_SCHEME	the signing scheme of <i>auth</i> is not supported by the TPM
TPM_RC_SIGNATURE	the signature is not genuine
TPM_RC_SIZE	input <i>cpHash</i> has wrong size
TPM_RC_VALUE	input <i>policyID</i> or <i>expiration</i> does not match the internal data in policy session

```

4  TPM_RC
5  TPM2_PolicySigned(
6      PolicySigned_In          *in,          // IN: input parameter list
7      PolicySigned_Out        *out          // OUT: output parameter list
8  )
9  {
10     TPM_RC          result = TPM_RC_SUCCESS;
11     SESSION         *session;
12     OBJECT          *authObject;
13     TPM2B_NAME      entityName;
14     TPM2B_DIGEST    authHash;
15     HASH_STATE      hashState;
16
17     // Input Validation
18
19     // Set up local pointers
20     session = SessionGet(in->policySession); // the session structure
21     authObject = ObjectGet(in->authObject); // pointer for the object
22                                             // providing authorization
23                                             // signature
24
25     //Only do input validation if this is not a trial policy session
26     if(session->attributes.isTrialPolicy == CLEAR)
27     {
28         // The object to validate the signature must be a signing key.
29         if( authObject->publicArea.objectAttributes.sign == CLEAR)
30             return TPM_RC_KEY + RC_PolicySigned_authObject;
31
32         // If it doesn't have a sensitive area loaded
33         // then it can't be a keyed hash signing key
34         if( authObject->publicArea.type == TPM_ALG_KEYEDHASH
35            && authObject->attributes.publicOnly == SET
36            )
37             return TPM_RC_HANDLE + RC_PolicySigned_authObject;
38
39         // Validate input 'noncePolicy'.
40         result = ValidatePolicyID(&in->nonceTPM, session);
41         if(result != TPM_RC_SUCCESS)

```

```

42     return TPM_RC_NONCE + RC_PolicySigned_nonceTPM;
43
44 // Validate input expiration. A TPM_RC_EXPIRED, TPM_RC_NV_UNAVAILABLE, or
45 // TPM_RC_NV_RATE error may be returned at this point
46 result = ValidateExpiration(in->expiration, session);
47 if(result != TPM_RC_SUCCESS)
48     return RcSafeAddToResult(result, RC_PolicySigned_expiration);
49
50 // A new cpHash is given in input parameter, but cpHash in session context
51 // is not empty, or is not the same as the new cpHash
52 if(    in->cpHashA.t.size != 0
53     && session->ul.cpHash.t.size != 0
54     && !Memory2BEqual(&in->cpHashA.b, &session->ul.cpHash.b)
55     )
56     return TPM_RC_CPHASH;
57
58 // A valid cpHash must have the same size as session hash digest
59 if(    in->cpHashA.t.size != 0
60     && in->cpHashA.t.size != CryptGetHashDigestSize(session->authHashAlg)
61     )
62     return TPM_RC_SIZE + RC_PolicySigned_cpHashA;
63
64 // Re-compute the digest being signed
65 /*(See part 3 specification)
66 // The digest is computed as:
67 //     aHash := hash ( nonceTPM | expiration | cpHashA | policyRef)
68 // where:
69 //     hash()     the hash associated with the signed auth
70 //     nonceTPM   the nonceTPM value from the TPM2_StartAuthSession .
71 //               response If the authorization is not limited to this
72 //               session, the size of this value is zero.
73 //     expiration time limit on authorization set by authorizing object.
74 //               This 32-bit value is set to zero if the expiration
75 //               time is not being set.
76 //     cpHashA   hash of the command parameters for the command being
77 //               approved using the hash algorithm of the PSAP session.
78 //               Set to NULLauth if the authorization is not limited
79 //               to a specific command.
80 //     policyRef hash of an opaque value determined by the authorizing
81 //               object. Set to the NULLdigest if no hash is present.
82 */
83 // Start hash
84 authHash.t.size = CryptStartHash(CryptGetSignHashAlg(&in->auth),
85                                 &hashState);
86
87 // add nonceTPM
88 CryptUpdateDigest2B(&hashState, &in->nonceTPM.b);
89
90 // add expiration
91 CryptUpdateDigestInt(&hashState, sizeof(UINT32), (BYTE*) &in->expiration);
92
93 // add cpHashA
94 CryptUpdateDigest2B(&hashState, &in->cpHashA.b);
95
96 // add policyRef
97 CryptUpdateDigest2B(&hashState, &in->policyRef.b);
98
99 // Complete digest
100 CryptCompleteHash2B(&hashState, &authHash.b);
101
102 // Validate Signature. A TPM_RC_SCHEME, TPM_RC_TYPE or TPM_RC_SIGNATURE
103 // error may be returned at this point
104 result = CryptVerifySignature(in->authObject, &authHash, &in->auth);
105 if(result != TPM_RC_SUCCESS)

```

```

106         return RcSafeAddToResult(result, RC_PolicySigned_auth);
107
108 // Internal Data Update
109 // Note that these values are not updated if the session is a trial session
110 // Update cpHash in policy session
111 if(in->cpHashA.t.size != 0)
112     session->ul.cpHash = in->cpHashA;
113
114 // Update expiration time in the policy session
115 if(in->expiration != 0)
116     UpdateTimeout((UINT64) in->expiration * 1000 + session->startTime,
117                 session);
118 }
119
120 // Update policy with input policyRef and name of auth key
121 // These values are updated even if the session is a trial session
122 entityName.t.size = EntityGetName(in->authObject, entityName.t.name);
123 PolicyUpdate(TPM_CC_PolicySigned, &entityName, &in->policyRef, session);
124
125 // Command Output
126
127 // Create ticket and timeout buffer if in->expiration != 0 and nonceTPM is
128 // null and this is not a trial session
129 if( in->expiration != 0
130     && in->nonceTPM.t.size == 0
131     && session->attributes.isTrialPolicy == CLEAR
132 )
133 {
134     UINT64 authTimeOut;
135     // Generate timeout buffer. The format of output timeout buffer is
136     // TPM-specific. In this implementation, we simply copy the value of
137     // timeout to the output buffer
138     authTimeOut = (UINT64) in->expiration * 1000 + session->startTime;
139     out->timeout.t.size = sizeof(UINT64);
140     UINT64_TO_BYTE_ARRAY(authTimeOut, out->timeout.t.buffer);
141
142     // Compute policy ticket
143     TicketComputeAuth(TPM_ST_AUTH_SIGNED, EntityGetHierarchy(in->authObject),
144                     authTimeOut, &in->cpHashA, &in->policyRef, &entityName,
145                     &out->policyTicket);
146 }
147 else
148 {
149     // Generate a null ticket.
150     // timeout buffer is null
151     out->timeout.t.size = 0;
152
153     // auth ticket is null
154     out->policyTicket.tag = TPM_ST_AUTH_SIGNED;
155     out->policyTicket.hierarchy = TPM_RH_NULL;
156     out->policyTicket.digest.t.size = 0;
157 }
158
159 return TPM_RC_SUCCESS;
160 }

```

## 25.4 TPM2\_PolicySecret

### 25.4.1 General Description

This command includes a secret-based authorization to a policy. The caller proves knowledge of the secret value using either a password or an HMAC-based authorization session.

The secret is the *authValue* of *authObject*, which may be any TPM entity with a handle and an associated *authValue*. This includes the reserved handles (for example, Platform, Storage, and Endorsement), NV Indexes, and loaded objects.

NOTE 1 The authorization value for a hierarchy cannot be used in this command if the hierarchy is disabled.

If the authorization check fails, then the normal dictionary attack logic is invoked.

If the authorization provided by the authorization session is valid, the command parameters are checked as described in 25.2.2.

When all validations have succeeded, *policySession*→*policyDigest* is updated by **PolicyUpdate()** (see 25.2.3).

**PolicyUpdate**(TPM\_CC\_PolicySecret, *authObject*→*Name*, *policyRef*) (15)

If the *cpHashA* command parameter is not an Empty Buffer, it is copied to *cpHash* in the session context.

The TPM will optionally produce a ticket as described in 25.2.5.

If the session is a trial session, *policySession*→*policyDigest* is updated as if the authorization is valid but no check is performed.

NOTE 2 If an HMAC is used to convey the authorization, a separate session is needed for the authorization. Because the HMAC in that authorization will include a nonce that prevents replay of the authorization, the value of the *nonceTPM* parameter in this command is limited. It is retained mostly to provide processing consistency with TPM2\_PolicySigned().

## 25.4.2 Command and Response

Table 115 — TPM2\_PolicySecret Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	see clause 8
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicySecret
TPMI_DH_ENTITY+	@authHandle	handle for an entity providing the authorization Auth Index: 1 Auth Role: USER
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None
TPM2B_NONCE	nonceTPM	the policy nonce for the session If the nonce is not included in the authorization qualification, this field is the Empty Buffer.
TPM2B_DIGEST	cpHashA	digest of the command parameters to which this authorization is limited This not the <i>cpHash</i> for this command but the <i>cpHash</i> for the command to which this policy session will be applied. If it is not limited, the parameter will be the Empty Buffer.
TPM2B_NONCE	policyRef	a reference to a policy relating to the authorization – may be the Empty Buffer Size is limited to be no larger than the nonce size supported on the TPM.
UINT32	expiration	time when authorization will expire, measured in seconds from the time that <i>nonceTPM</i> was generated If <i>expiration</i> is zero, a NULL Ticket is returned.

Table 116 — TPM2\_PolicySecret Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_TIMEOUT	timeout	implementation-specific time value used to indicate to the TPM when the ticket expires; this ticket will use the TPMT_ST_AUTH_SECRET structure tag
TPMT_TK_AUTH	policyTicket	produced if the command succeeds and <i>expiration</i> in the command was non-zero

## 25.4.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PolicySecret_fp.h"
3  #include "Policy_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_CPHASH	<i>cpHash</i> for policy was previously set to a value that is not the same as <i>cpHashA</i>
TPM_RC_EXPIRED	<i>expiration</i> indicates a time in the past
TPM_RC_NONCE	<i>nonceTPM</i> does not match the nonce associated with <i>policySession</i>
TPM_RC_SIZE	<i>cpHashA</i> is not the size of a digest for the hash associated with <i>policySession</i>
TPM_RC_VALUE	input <i>policyID</i> or expiration does not match the internal data in policy session

```

4  TPM_RC
5  TPM2_PolicySecret(
6      PolicySecret_In          *in,    // IN: input parameter list
7      PolicySecret_Out        *out     // OUT: output parameter list
8  )
9  {
10     TPM_RC                    result;
11     SESSION                   *session;
12     TPM2B_NAME                entityName;
13
14     // Input Validation
15
16     // Get pointer to the session structure
17     session = SessionGet(in->policySession);
18
19     //Only do input validation if this is not a trial policy session
20     if(session->attributes.isTrialPolicy == CLEAR)
21     {
22
23         // Validate input policyID. A TPM_RC_VALUE error may be returned at
24         result = ValidatePolicyID(&in->nonceTPM, session);
25         if(result != TPM_RC_SUCCESS)
26             return TPM_RC_NONCE + RC_PolicySecret_nonceTPM;
27
28         // Validate input expiration. A TPM_RC_EXPIRED error may be returned at
29         // this point
30         result = ValidateExpiration(in->expiration, session);
31         if(result != TPM_RC_SUCCESS)
32             return TPM_RC_EXPIRED + RC_PolicySecret_expiration;
33
34         // A new cpHash is given in input parameter, but cpHash in session context
35         // is not empty, or is not the same as the new cpHash
36         if( in->cpHashA.t.size != 0
37            && session->ul.cpHash.t.size != 0
38            && !Memory2BEqual(&in->cpHashA.b, &session->ul.cpHash.b))
39             return TPM_RC_CPHASH;
40
41         // A valid cpHash must have the same size as session hash digest
42         if( in->cpHashA.t.size != 0
43            && in->cpHashA.t.size != CryptGetHashDigestSize(session->authHashAlg))
44             return TPM_RC_SIZE + RC_PolicySecret_cpHashA;
45
46         // Internal Data Update
47         // Update cpHashA

```

```

48     // Note that these value are updated only if the session is not a
49     // trial session
50     if(in->cpHashA.t.size != 0)
51     {
52         session->ul.cpHash = in->cpHashA;
53     }
54
55     // Update expiration time
56     if(in->expiration != 0)
57         UpdateTimeout((UINT64) in->expiration * 1000 + session->startTime,
58                     session);
59 }
60
61 // Update policy with input policyRef and name of auth key
62 // This value is computed even for trial sessions
63 entityName.t.size = EntityGetName(in->authHandle, entityName.t.name);
64 PolicyUpdate(TPM_CC_PolicySecret, &entityName, &in->policyRef, session);
65
66 // Command Output
67
68 // Create ticket and timeout buffer if in->expiration != 0 and nonceTPM is
69 // null and this is not a trial session.
70 if( in->expiration != 0
71     && in->nonceTPM.t.size == 0
72     && session->attributes.isTrialPolicy == CLEAR
73 )
74 {
75     UINT64     authTimeOut;
76     // Generate timeout buffer. The format of output timeout buffer is
77     // TPM-specific. In this implementation, we simply copy the value of
78     // timeout to the output buffer
79     authTimeOut = (UINT64) in->expiration * 1000 + session->startTime;
80     out->timeout.t.size = sizeof(UINT64);
81     UINT64_TO_BYTE_ARRAY(authTimeOut, out->timeout.t.buffer);
82
83     // Compute policy ticket
84     TicketComputeAuth(TPM_ST_AUTH_SECRET, EntityGetHierarchy(in->authHandle),
85                     authTimeOut, &in->cpHashA, &in->policyRef,
86                     &entityName, &out->policyTicket);
87 }
88 else
89 {
90     // timeout buffer is null
91     out->timeout.t.size = 0;
92
93     // auth ticket is null
94     out->policyTicket.tag = TPM_ST_AUTH_SECRET;
95     out->policyTicket.hierarchy = TPM_RH_NULL;
96     out->policyTicket.digest.t.size = 0;
97 }
98
99 return TPM_RC_SUCCESS;
100 }

```



## 25.5 TPM2\_PolicyTicket

### 25.5.1 General Description

This command is similar to TPM2\_PolicySigned() except that it takes a ticket instead of a signed authorization. The ticket represents a validated authorization that had an expiration time associated with it.

The parameters of this command are checked as described in 25.2.2.

If the checks succeed, the TPM uses the *timeout*, *cpHashA*, *policyRef*, and *keyName* to construct a ticket to compare with the value in *ticket*. If these tickets match, then the TPM will create a TPM2B\_NAME (*objectName*) using *authName* and update the context of *policySession* by **PolicyUpdate()** (see 25.2.3).

**PolicyUpdate**(*commandCode*, *authName*, *policyRef*) (16)

If the structure tag of ticket is TPM\_ST\_AUTH\_SECRET, then *commandCode* will be TPM\_CC\_PolicySecret. If the structure tag of ticket is TPM\_ST\_AUTH\_SIGNED, then *commandCode* will be TPM\_CC\_PolicySigned.

If the *cpHashA* command parameter is not an Empty Buffer, it may be copied to *cpHash* in the session context as described in 25.2.1.

## 25.5.2 Command and Response

Table 117 — TPM2\_PolicyTicket Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	see clause 8
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyTicket
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None
TPM2B_TIMEOUT	timeout	time when authorization will expire The contents are TPM specific. This shall be the value returned when ticket was produced.
TPM2B_DIGEST	cpHashA	digest of the command parameters to which this authorization is limited If it is not limited, the parameter will be the Empty Buffer.
TPM2B_NONCE	policyRef	reference to a qualifier for the policy – may be the Empty Buffer
TPM2B_NAME	authName	name of the object that provided the authorization
TPMT_TK_AUTH	ticket	an authorization ticket returned by the TPM in response to a TPM2_PolicySigned() or TPM2_PolicySecret()

Table 118 — TPM2\_PolicyTicket Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 25.5.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PolicyTicket_fp.h"
3  #include "Policy_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_CPHASH	policy's <i>cpHash</i> was previously set to a different value
TPM_RC_EXPIRED	<i>timeout</i> value in the ticket is in the past and the ticket has expired
TPM_RC_SIZE	<i>timeout</i> or <i>cpHash</i> has invalid size for the
TPM_RC_TICKET	<i>ticket</i> is not valid

```

4  TPM_RC
5  TPM2_PolicyTicket(
6      PolicyTicket_In      *in      // IN: input parameter list
7  )
8  {
9      TPM_RC      result;
10     SESSION      *session;
11     UINT64      timeout;
12     TPMT_TK_AUTH      ticketToCompare;
13     TPM_CC      commandCode = TPM_CC_PolicySecret;
14
15     // Input Validation
16
17     // Get pointer to the session structure
18     session = SessionGet(in->policySession);
19
20     // NOTE: There is no check for a trial policy session. Tickets are
21     // not created in a trial policy session because no data has been validated
22
23     // A new cpHash is given in input parameter, but cpHash in session context
24     // is not empty, or is not the same as the new cpHash
25     if( in->cpHashA.t.size != 0
26         && session->ul.cpHash.t.size != 0
27         && !Memory2BEqual(&in->cpHashA.b, &session->ul.cpHash.b))
28         return TPM_RC_CPHASH;
29
30     // A valid cpHash must have the same size as session hash digest
31     if( in->cpHashA.t.size != 0
32         && in->cpHashA.t.size != CryptGetHashDigestSize(session->authHashAlg))
33         return TPM_RC_SIZE + RC_PolicyTicket_cpHashA;
34
35     // Restore timeout data. The format of timeout buffer is TPM-specific.
36     // In this implementation, we simply copy the value of timeout to the
37     // buffer.
38     if(in->timeout.t.size != sizeof(UINT64))
39         return TPM_RC_SIZE + RC_PolicyTicket_timeout;
40
41     // Cannot compare time if clock stop advancing. A TPM_RC_NV_UNAVAILABLE
42     // or TPM_RC_NV_RATE error may be returned here.
43     result = NvIsAvailable();
44     if(result != TPM_RC_SUCCESS)
45         return result;
46
47     timeout = BYTE_ARRAY_TO_UINT64(in->timeout.t.buffer);
48     if(timeout < go.clock)
49         return TPM_RC_EXPIRED + RC_PolicyTicket_timeout;

```

```
50
51 // Validate Ticket
52 // Re-generate policy ticket by input parameters
53 TicketComputeAuth(in->ticket.tag, in->ticket.hierarchy, timeout, &in->cpHashA,
54                  &in->policyRef, &in->authName, &ticketToCompare);
55
56 // Compare generated digest with input ticket digest
57 if(!Memory2BEqual(&in->ticket.digest.b, &ticketToCompare.digest.b))
58     return TPM_RC_TICKET + RC_PolicyTicket_ticket;
59
60 // If the ticket is valid, update session timeout.
61 UpdateTimeout(timeout, session);
62
63 // Internal Data Update
64
65 // Update policy with input policyRef and name of auth key
66 if(in->ticket.tag == TPM_ST_AUTH_SIGNED)
67     commandCode = TPM_CC_PolicySigned;
68 else if(in->ticket.tag == TPM_ST_AUTH_SECRET)
69     commandCode = TPM_CC_PolicySecret;
70 else
71     // There could only be two possible tag values. Any other value should
72     // be caught by the ticket validation process.
73     pAssert(FALSE);
74 PolicyUpdate(commandCode, &in->authName, &in->policyRef, session);
75
76 // if cpHash was specified, update the policy context
77 if(in->cpHashA.t.size != 0)
78     session->u1.cpHash = in->cpHashA;
79
80 return TPM_RC_SUCCESS;
81 }
```

## 25.6 TPM2\_PolicyOR

### 25.6.1 General Description

This command allows options in authorizations without requiring that the TPM evaluate all of the options. If a policy may be satisfied by different sets of conditions, the TPM need only evaluate one set that satisfies the policy. This command will indicate that one of the required sets of conditions has been satisfied.

*PolicySession*→*policyDigest* is compared against the list of provided values. If the current *policySession*→*policyDigest* does not match any value in the list, the TPM shall return TPM\_RC\_VALUE. Otherwise, it will replace *policySession*→*policyDigest* with the digest of the concatenation of all of the digests and return TPM\_RC\_SUCCESS.

If *policySession* is a trial session, the TPM will assume that *policySession*→*policyDigest* matches one of the list entries and compute the new value of *policyDigest*.

The algorithm for computing the new value for *policyDigest* of *policySession* is:

- a) Concatenate all the digest values in *pHashList*:

$$digests := pHashList.digests[1].buffer || \dots || pHashList.digests[n].buffer \quad (17)$$

NOTE 1 The TPM makes no check to see if the size of an entry matches the size of the digest of the policy.

- b) Reset *policyDigest* to a Zero Digest.

- c) Extend the command code and the hashes computed in step a) above:

$$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM\_CC\_PolicyOR || digests) \quad (18)$$

NOTE 2 The computation in b) and c) above is equivalent to:

$$policyDigest_{new} := H_{policyAlg}(0\dots0 || TPM\_CC\_PolicyOR || digests)$$

A TPM shall support a list with at least eight tagged digest values.

NOTE 3 If policies are to be portable between TPMs, then they should not use more than eight values.

## 25.6.2 Command and Response

Table 119 — TPM2\_PolicyOR Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyOR.
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None
TPML_DIGEST	pHashList	the list of hashes to check for a match

Table 120 — TPM2\_PolicyOR Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 25.6.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PolicyOR_fp.h"
3  #include "Policy_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_VALUE	no digest in <i>pHashList</i> matched the current value of <i>policyDigest</i> for <i>policySession</i>

```

4  TPM_RC
5  TPM2_PolicyOR(
6      PolicyOR_In *in          // IN: input parameter list
7  )
8  {
9      SESSION      *session;
10     UINT32       i;
11
12     // Input Validation and Update
13
14     // Get pointer to the session structure
15     session = SessionGet(in->policySession);
16
17     // Compare and Update Internal Session policy if match
18     for(i = 0; i < in->pHashList.count; i++)
19     {
20         if( session->attributes.isTrialPolicy == SET
21            || (Memory2BEqual(&session->u2.policyDigest.b,
22                            &in->pHashList.digests[i].b))
23            )
24         {
25             // Found a match
26             HASH_STATE      hashState;
27             TPM_CC          commandCode = TPM_CC_PolicyOR;
28
29             // Start hash
30             session->u2.policyDigest.t.size = CryptStartHash(session->authHashAlg,
31                                                            &hashState);
32             // Set policyDigest to 0 string and add it to hash
33             MemorySet(session->u2.policyDigest.t.buffer, 0,
34                      session->u2.policyDigest.t.size);
35             CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
36
37             // add command code
38             CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
39
40             // Add each of the hashes in the list
41             for(i = 0; i < in->pHashList.count; i++)
42             {
43                 // Extend policyDigest
44                 CryptUpdateDigest2B(&hashState, &in->pHashList.digests[i].b);
45             }
46             // Complete digest
47             CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
48
49             return TPM_RC_SUCCESS;
50         }
51     }
52     // None of the values in the list matched the current policyDigest
53     return TPM_RC_VALUE + RC_PolicyOR_pHashList;
54 }

```

## 25.7 TPM2\_PolicyPCR

### 25.7.1 General Description

This command is used to cause conditional gating of a policy based on PCR. This allows one group of authorizations to occur when PCR are in one state and a different set of authorizations when the PCR are in a different state. If this command is used for a trial *policySession*, *policySession*→*policyDigest* will be updated using the values from the command rather than the values from digest of the TPM PCR.

The TPM will modify the *pcrs* parameter so that bits that correspond to unimplemented PCR are CLEAR. If *policySession* is not a trial policy session, the TPM will use the modified value of *pcrs* to select PCR values to hash according to Part 1, *Selecting Multiple PCR*. The hash algorithm of the policy session is used to compute a digest (*digestTPM*) of the selected PCR. If *pcrDigest* does not have a length of zero, then it is compared to *digestTPM*; and if the values do not match, the TPM shall return TPM\_RC\_VALUE and make no change to *policySession*→*policyDigest*. If the values match, or if the length of *pcrDigest* is zero, then *policySession*→*policyDigest* is extended by:

$$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM\_CC\_PolicyPCR || pcrs || digestTPM) \quad (19)$$

where

<i>pcrs</i>	the <i>pcrs</i> parameter with bits corresponding to unimplemented PCR set to 0
<i>digestTPM</i>	the digest of the selected PCR using the hash algorithm of the policy session

NOTE 1 If the caller provides the expected PCR value, the intention is that the policy evaluation stop at that point if the PCR do not match. If the caller does not provide the expected PCR value, then the validity of the settings will not be determined until an attempt is made to use the policy for authorization. If the policy is constructed such that the PCR check comes before user authorization checks, this early termination would allow software to avoid unnecessary prompts for user input to satisfy a policy that would fail later due to incorrect PCR values.

After this command completes successfully, the TPM shall return TPM\_RC\_PCR\_CHANGED if the policy session is used for authorization and the PCR are not known to be correct.

The TPM uses a “generation” number (*pcrUpdateCounter*) that is incremented each time PCR are updated (unless the PCR being changed is specified not to cause a change to this counter). The value of this counter is stored in the policy session context (*policySession*→*pcrUpdateCounter*) when this command is executed. When the policy is used for authorization, the current value of the counter is compared to the value in the policy session context and the authorization will fail if the values are not the same.

When this command is executed, *policySession*→*pcrUpdateCounter* is checked to see if it has been previously set (in the reference implementation, it has a value of zero if not previously set). If it has been set, it will be compared with the current value of *pcrUpdateCounter* to determine if any PCR changes have occurred. If the values are different, the TPM shall return TPM\_RC\_PCR\_CHANGED. If *policySession*→*pcrUpdateCounter* has not been set, then it is set to the current value of *pcrUpdateCounter*.

If *policySession* is a trial policy session, the TPM will not check any PCR and will compute:

$$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM\_CC\_PolicyPCR || pcrs || pcrDigest) \quad (20)$$

In this computation, *pcrs* is the input parameter without modification.

NOTE 2 The *pcrs* parameter is expected to match the configuration of the TPM for which the policy is being computed which may not be the same as the TPM on which the trial policy is being computed.



25.7.2 Command and Response

**Table 121 — TPM2\_PolicyPCR Command**

Type	Name	Description
TPML_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyPCR
TPML_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None
TPM2B_DIGEST	pcrDigest	expected digest value of the selected PCR using the hash algorithm of the session; may be zero length
TPML_PCR_SELECTION	pcrs	the PCR to include in the check digest

**Table 122 — TPM2\_PolicyPCR Response**

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 25.7.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PolicyPCR_fp.h"

```

Error Returns	Meaning
TPM_RC_VALUE	if provided, <i>pcrDigest</i> does not match the current PCR settings
TPM_RC_PCR_CHANGED	a previous TPM2_PolicyPCR() set <i>pcrCounter</i> and it has changed

```

3  TPM_RC
4  TPM2_PolicyPCR(
5      PolicyPCR_In    *in          // IN: input parameter list
6  )
7  {
8      SESSION          *session;
9      TPM2B_DIGEST     pcrDigest;
10     BYTE              pcrcs[sizeof(TPML_PCR_SELECTION)];
11     UINT32            pcrSize;
12     BYTE              *buffer;
13     TPM_CC             commandCode = TPM_CC_PolicyPCR;
14     HASH_STATE        hashState;
15
16     // Input Validation
17
18     // Get pointer to the session structure
19     session = SessionGet(in->policySession);
20
21     // Do validation for non trial session
22     if(session->attributes.isTrialPolicy == CLEAR)
23     {
24         // Make sure that this is not going to invalidate a previous PCR check
25         if(session->pcrCounter != 0 && session->pcrCounter != gr.pcrCounter)
26             return TPM_RC_PCR_CHANGED;
27
28         // Compute current PCR digest
29         PCRComputeCurrentDigest(session->authHashAlg, &in->pcrcs, &pcrDigest);
30
31         // If the caller specified the PCR digest and it does not
32         // match the current PCR settings, return an error..
33         if(in->pcrDigest.t.size != 0)
34         {
35             if(!Memory2BEqual(&in->pcrDigest.b, &pcrDigest.b))
36                 return TPM_RC_VALUE + RC_PolicyPCR_pcrDigest;
37         }
38     }
39     else
40     {
41         // For trial session, just use the input PCR digest
42         pcrDigest = in->pcrDigest;
43     }
44     // Internal Data Update
45
46     // Update policy hash
47     // policyDigestnew = hash( policyDigestold || TPM_CC_PolicyPCR
48     //                          || pcrcs || pcrDigest)
49     // Start hash
50     CryptStartHash(session->authHashAlg, &hashState);
51
52     // add old digest, which may be empty
53     CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

```

```
54
55     // add commandCode
56     CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
57
58     // add PCRS
59     buffer = pcrs;
60     pcrSize = TPML_PCR_SELECTION_Marshal(&in->pcrs, &buffer, NULL);
61     CryptUpdateDigest(&hashState, pcrSize, pcrs);
62
63     // add PCR digest
64     CryptUpdateDigest2B(&hashState, &pcrDigest.b);
65
66     // complete the hash and get the results
67     CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
68
69     // update pcrCounter in session context for non trial session
70     if(session->attributes.isTrialPolicy == CLEAR)
71     {
72         session->pcrCounter = gr.pcrCounter;
73     }
74
75     return TPM_RC_SUCCESS;
76 }
```

## 25.8 TPM2\_PolicyLocality

### 25.8.1 General Description

This command indicates that the authorization will be limited to a specific locality.

*policySession*→*commandLocality* is a parameter kept in the session context. It is initialized when the policy session is started to allow the policy to apply to any locality.

If *locality* has a value greater than 31, then an extended locality is indicated. For an extended locality, the TPM will validate that *policySession*→*commandLocality* has not previously been set or that the current value of *policySession*→*commandLocality* is the same as *locality* (TPM\_RC\_RANGE).

When *locality* is not an extended locality, the TPM will validate that the *policySession*→*commandLocality* is not set or is not set to an extended locality value (TPM\_RC\_RANGE). If not the TPM will disable any locality not SET in the *locality* parameter. If the result of disabling localities results in no locality being enabled, the TPM will return TPM\_RC\_RANGE.

If no error occurred in the validation of *locality*, *policySession*→*policyDigest* is extended with

$$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM\_CC\_PolicyLocality || locality) \quad (21)$$

Then *policySession*→*commandLocality* is updated to indicate which localities are still allowed after execution of TPM2\_PolicyLocality().

When the policy session is used to authorize a command, the authorization will fail if the locality used for the command is not one of the enabled localities in *policySession*→*commandLocality*.

25.8.2 Command and Response

**Table 123 — TPM2\_PolicyLocality Command**

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyLocality
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None
TPMA_LOCALITY	locality	the allowed localities for the policy

**Table 124 — TPM2\_PolicyLocality Response**

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 25.8.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PolicyLocality_fp.h"

```

Limit a policy to a specific locality

Error Returns	Meaning
TPM_RC_RANGE	all the locality values selected by <i>locality</i> have been disabled by previous TPM2_PolicyLocality() calls.

```

3  TPM_RC
4  TPM2_PolicyLocality(
5      PolicyLocality_In  *in          // IN: input parameter list
6  )
7  {
8      SESSION      *session;
9      BYTE          marshalBuffer[sizeof(TPMA_LOCALITY)];
10     BYTE          prevSetting[sizeof(TPMA_LOCALITY)];
11     UINT32        marshalSize;
12     BYTE          *buffer;
13     TPM_CC        commandCode = TPM_CC_PolicyLocality;
14     HASH_STATE    hashState;
15
16     // Input Validation
17
18     // Get pointer to the session structure
19     session = SessionGet(in->policySession);
20
21     // Get new locality setting in canonical form
22     buffer = marshalBuffer;
23     marshalSize = TPMA_LOCALITY_Marshal(&in->locality, &buffer, NULL);
24
25     // Its an error if the locality parameter is zero
26     if(marshalBuffer[0] == 0)
27         return TPM_RC_RANGE + RC_PolicyLocality_locality;
28
29     // Get existing locality setting in canonical form
30     buffer = prevSetting;
31     TPMA_LOCALITY_Marshal(&session->commandLocality, &buffer, NULL);
32
33     // If the locality has been previously set, then it needs to be the same
34     // type as the input locality (i.e. both extended or both normal
35     if(prevSetting[0] != 0 && ((prevSetting[0] <= 0) != (marshalBuffer[0] <= 0)))
36         return TPM_RC_RANGE + RC_PolicyLocality_locality;
37
38
39     // See if the input is a regular or extended locality
40     if(marshalBuffer[0] < 32)
41     {
42         // For regular locality
43         // The previous setting must not be an extended locality
44         if(prevSetting[0] > 31)
45             return TPM_RC_RANGE + RC_PolicyLocality_locality;
46
47         // if there was no previous setting, start with all normal localities
48         // enabled
49         if(prevSetting[0] == 0)
50             prevSetting[0] = 0x1F;
51
52         // AND the new setting with the previous setting and store it in prevSetting

```

```
53     prevSetting[0] &= marshalBuffer[0];
54
55     // The result setting can not be 0
56     if(prevSetting[0] == 0)
57         return TPM_RC_RANGE + RC_PolicyLocality_locality;
58     }
59     else
60     {
61         // for extended locality
62         // if the locality has already been set, then it must match the
63         if(prevSetting[0] != 0 && prevSetting[0] != marshalBuffer[0])
64             return TPM_RC_RANGE + RC_PolicyLocality_locality;
65
66         // Setting is OK
67         prevSetting[0] = marshalBuffer[0];
68     }
69 }
70
71 // Internal Data Update
72
73 // Update policy hash
74 // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyLocality || locality)
75 // Start hash
76 CryptStartHash(session->authHashAlg, &hashState);
77
78 // add old digest, which may be empty
79 CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
80
81 // add commandCode
82 CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
83
84 // add input locality
85 CryptUpdateDigest(&hashState, marshalSize, marshalBuffer);
86
87 // complete the digest
88 CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
89
90 // update session locality by unmarshal function. The function must succeed
91 // because both input and existing locality setting have been validated.
92 buffer = prevSetting;
93 TPMA_LOCALITY_Unmarshal(&session->commandLocality, &buffer,
94                        (INT32 *) &marshalSize);
95
96 return TPM_RC_SUCCESS;
97 }
```

## 25.9 TPM2\_PolicyNV

### 25.9.1 General Description

This command is used to cause conditional gating of a policy based on the contents of an NV Index.

If *policySession* is a trial policy session, the TPM will update *policySession*→*policyDigest* as shown in equations (22) and (23) below and return TPM\_RC\_SUCCESS. It will not perform any validation. The remainder of this general description would apply only if *policySession* is not a trial policy session.

An authorization session providing authorization to read the NV Index shall be provided.

NOTE 1 If read access is controlled by policy, the policy should include a branch that authorizes a TPM2\_PolicyNV().

If TPMA\_NV\_WRITTEN is not SET in the NV Index, the TPM shall return TPM\_RC\_NV\_UNINITIALIZED.

The TPM will validate that the size of *operandB* plus offset is not greater than the size of the NV Index. If it is, the TPM shall return TPM\_RC\_SIZE.

The TPM will perform the indicated arithmetic check on the indicated portion of the selected NV Index. If the check fails, the TPM shall return TPM\_RC\_POLICY and not change *policySession*→*policyDigest*. If the check succeeds, the TPM will hash the arguments:

$$args := H_{policyAlg}(operand.buffer || offset || operation) \quad (22)$$

where

$H_{policyAlg}()$	hash function using the algorithm of the policy session
<i>operandB</i>	the value used for the comparison
<i>offset</i>	offset from the start of the NV Index data to start the comparison
<i>operation</i>	the operation parameter indicating the comparison being performed

The value of *args* and the Name of the NV Index are extended to *policySession*→*policyDigest* by

$$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM\_CC\_PolicyNV || args || nvIndex \rightarrow Name) \quad (23)$$

where

$H_{policyAlg}()$	hash function using the algorithm of the policy session
<i>args</i>	value computed in equation (22)
<i>nvIndex</i> → <i>Name</i>	the Name of the NV Index

The signed arithmetic operations are performed using twos-compliment.

Magnitude comparisons assume that the octet at offset zero in the referenced NV location and in *operandB* contain the most significant octet of the data.

NOTE 2 When an Index is written, it has a different authorization name than an Index that has not been written. It is possible to use this change in the NV Index to create a write-once Index.



25.9.2 Command and Response

**Table 125 — TPM2\_PolicyNV Command**

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyNV
TPMI_RH_NV_AUTH	@authHandle	handle indicating the source of the authorization value Auth Index: 1 Auth Role: USER
TPMI_RH_NV_INDEX	nvIndex	the NV Index of the area to read Auth Index: None
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None
TPM2B_OPERAND	operandB	the second operand
UINT16	offset	the offset in the NV Index for the start of operand A
TPM_EO	operation	the comparison to make

**Table 126 — TPM2\_PolicyNV Response**

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 25.9.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PolicyNV_fp.h"
3  #include "Policy_spt_fp.h"
4  #include "NV_spt_fp.h"           // Include NV support routine for read access check

```

Error Returns	Meaning
TPM_RC_AUTH_TYPE	NV index authorization type is not correct
TPM_RC_NV_LOCKED	NV index read locked
TPM_RC_NV_UNINITIALIZED	the NV index has not been initialized
TPM_RC_POLICY	the comparison to the NV contents failed
TPM_RC_SIZE	the size of <i>nvIndex</i> data starting at <i>offset</i> is less than the size of <i>operandB</i>

```

5  TPM_RC
6  TPM2_PolicyNV(
7      PolicyNV_In      *in          // IN: input parameter list
8  )
9  {
10     TPM_RC          result;
11     SESSION        *session;
12     NV_INDEX       nvIndex;
13     BYTE           nvBuffer[sizeof(in->operandB.t.buffer)];
14     TPM2B_NAME     nvName;
15     TPM_CC         commandCode = TPM_CC_PolicyNV;
16     HASH_STATE     hashState;
17     TPM2B_DIGEST   argHash;
18
19     // Input Validation
20
21     // Get NV index information
22     NvGetIndexInfo(in->nvIndex, &nvIndex);
23
24     // Get pointer to the session structure
25     session = SessionGet(in->policySession);
26
27     //If this is a trial policy, skip all validations and the operation
28     if(session->attributes.isTrialPolicy == CLEAR)
29     {
30         // NV Read access check. NV index should be allowed for read. A
31         // TPM_RC_AUTH_TYPE or TPM_RC_NV_LOCKED error may be return at this
32         // point
33         result = NvReadAccessChecks(in->authHandle, in->nvIndex);
34         if(result != TPM_RC_SUCCESS) return result;
35
36         // Valid NV data size should not be smaller than input operandB size
37         if((nvIndex.publicArea.dataSize - in->offset) < in->operandB.t.size)
38             return TPM_RC_SIZE + RC_PolicyNV_operandB;
39
40         // Arithmetic Comparison
41
42         // Get NV data. The size of NV data equals the input operand B size
43         NvGetIndexData(in->nvIndex, &nvIndex, in->offset,
44             in->operandB.t.size, nvBuffer);
45
46         switch(in->operation)

```

```

47     {
48     case TPM_EO_EQ:
49         // compare A = B
50         if (CryptCompare(in->operandB.t.size, nvBuffer,
51                         in->operandB.t.size, in->operandB.t.buffer) != 0)
52             return TPM_RC_POLICY;
53         break;
54     case TPM_EO_NEQ:
55         // compare A != B
56         if (CryptCompare(in->operandB.t.size, nvBuffer,
57                         in->operandB.t.size, in->operandB.t.buffer) == 0)
58             return TPM_RC_POLICY;
59         break;
60     case TPM_EO_SIGNED_GT:
61         // compare A > B signed
62         if (CryptCompareSigned(in->operandB.t.size, nvBuffer,
63                               in->operandB.t.size, in->operandB.t.buffer) <= 0)
64             return TPM_RC_POLICY;
65         break;
66     case TPM_EO_UNSIGNED_GT:
67         // compare A > B unsigned
68         if (CryptCompare(in->operandB.t.size, nvBuffer,
69                         in->operandB.t.size, in->operandB.t.buffer) <= 0)
70             return TPM_RC_POLICY;
71         break;
72     case TPM_EO_SIGNED_LT:
73         // compare A < B signed
74         if (CryptCompareSigned(in->operandB.t.size, nvBuffer,
75                               in->operandB.t.size, in->operandB.t.buffer) >= 0)
76             return TPM_RC_POLICY;
77         break;
78     case TPM_EO_UNSIGNED_LT:
79         // compare A < B unsigned
80         if (CryptCompare(in->operandB.t.size, nvBuffer,
81                         in->operandB.t.size, in->operandB.t.buffer) >= 0)
82             return TPM_RC_POLICY;
83         break;
84     case TPM_EO_SIGNED_GE:
85         // compare A >= B signed
86         if (CryptCompareSigned(in->operandB.t.size, nvBuffer,
87                               in->operandB.t.size, in->operandB.t.buffer) < 0)
88             return TPM_RC_POLICY;
89         break;
90     case TPM_EO_UNSIGNED_GE:
91         // compare A >= B unsigned
92         if (CryptCompare(in->operandB.t.size, nvBuffer,
93                         in->operandB.t.size, in->operandB.t.buffer) < 0)
94             return TPM_RC_POLICY;
95         break;
96     case TPM_EO_SIGNED_LE:
97         // compare A <= B signed
98         if (CryptCompareSigned(in->operandB.t.size, nvBuffer,
99                               in->operandB.t.size, in->operandB.t.buffer) > 0)
100             return TPM_RC_POLICY;
101         break;
102     case TPM_EO_UNSIGNED_LE:
103         // compare A <= B unsigned
104         if (CryptCompare(in->operandB.t.size, nvBuffer,
105                         in->operandB.t.size, in->operandB.t.buffer) > 0)
106             return TPM_RC_POLICY;
107         break;
108     case TPM_EO_BITSET:
109         // All bits SET in B are SET in A. ((A&B)=B)
110     {

```

```

111         UINT32 i;
112         for (i = 0; i < in->operandB.t.size; i++)
113             if((nvBuffer[i] & in->operandB.t.buffer[i])
114                 != in->operandB.t.buffer[i])
115                 return TPM_RC_POLICY;
116     }
117     break;
118     case TPM_EO_BITCLEAR:
119         // All bits SET in B are CLEAR in A. ((A&B)=0)
120         {
121             UINT32 i;
122             for (i = 0; i < in->operandB.t.size; i++)
123                 if((nvBuffer[i] & in->operandB.t.buffer[i]) != 0)
124                     return TPM_RC_POLICY;
125         }
126     break;
127     default:
128         pAssert(FALSE);
129         break;
130     }
131 }
132
133 // Internal Data Update
134
135 // Start argument hash
136 argHash.t.size = CryptStartHash(session->authHashAlg, &hashState);
137
138 // add operandB
139 CryptUpdateDigest2B(&hashState, &in->operandB.b);
140
141 // add offset
142 CryptUpdateDigestInt(&hashState, sizeof(UINT16), &in->offset);
143
144 // add operation
145 CryptUpdateDigestInt(&hashState, sizeof(TPM_EO), &in->operation);
146
147 // complete argument digest
148 CryptCompleteHash2B(&hashState, &argHash.b);
149
150 // Update policyDigest
151 // Start digest
152 CryptStartHash(session->authHashAlg, &hashState);
153
154 // add old digest, which may be empty
155 CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
156
157 // add commandCode
158 CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
159
160 // add argument digest
161 CryptUpdateDigest2B(&hashState, &argHash.b);
162
163 // Adding nvName
164 nvName.t.size = EntityGetName(in->nvIndex, nvName.t.name);
165 CryptUpdateDigest2B(&hashState, &nvName.b);
166
167 // complete the digest
168 CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
169
170 return TPM_RC_SUCCESS;
171 }

```

## 25.10 TPM2\_PolicyCounterTimer

### 25.10.1 General Description

This command is used to cause conditional gating of a policy based on the contents of the TPMS\_TIME\_INFO structure.

If *policySession* is a trial policy session, the TPM will update *policySession*→*policyDigest* as shown in equations (24) and (25) below and return TPM\_RC\_SUCCESS. It will not perform any validation. The remainder of this general description would apply only if *policySession* is not a trial policy session.

The TPM will perform the indicated arithmetic check on the indicated portion of the TPMS\_TIME\_INFO structure. If the check fails, the TPM shall return TPM\_RC\_POLICY and not change *policySession*→*policyDigest*. If the check succeeds, the TPM will hash the arguments:

$$args := H_{policyAlg}(operandB.buffer || offset || operation) \quad (24)$$

where

$H_{policyAlg}()$	hash function using the algorithm of the policy session
<i>operandB.buffer</i>	the value used for the comparison
<i>offset</i>	offset from the start of the TPMS_TIME_INFO structure at which the comparison starts
<i>operation</i>	the operation parameter indicating the comparison being performed

The value of *args* is extended to *policySession*→*policyDigest* by

$$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM\_CC\_PolicyCounterTimer || args) \quad (25)$$

where

$H_{policyAlg}()$	hash function using the algorithm of the policy session
<i>args</i>	value computed in equation (24)

The signed arithmetic operations are performed using twos-compliment.

Magnitude comparisons assume that the octet at offset zero in the referenced location and in *operandB* contain the most significant octet of the data.

## 25.10.2 Command and Response

Table 127 — TPM2\_PolicyCounterTimer Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyCounterTimer
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None
TPM2B_OPERAND	operandB	the second operand
UINT16	offset	the offset in TPMS_TIME_INFO structure for the start of operand A
TPM_EO	operation	the comparison to make

Table 128 — TPM2\_PolicyCounterTimer Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 25.10.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PolicyCounterTimer_fp.h"
3  #include "Policy_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_POLICY	the comparison of the selected portion of the TPMS_TIME_INFO with <i>operandB</i> failed
TPM_RC_RANGE	<i>offset + size</i> exceed size of TPMS_TIME_INFO structure

```

4  TPM_RC
5  TPM2_PolicyCounterTimer(
6      PolicyCounterTimer_In  *in          // IN: input parameter list
7  )
8  {
9      TPM_RC          result;
10     SESSION         *session;
11     BYTE            infoData[sizeof(TPMS_TIME_INFO)]; // data buffer of
12                                                         // TPMS_TIME_INFO
13     TPM_CC          commandCode = TPM_CC_PolicyCounterTimer;
14     HASH_STATE      hashState;
15     TPM2B_DIGEST    argHash;
16
17     // Input Validation
18
19     // If the command is going to use any part of the counter or timer, need
20     // to verify that time is advancing.
21     // The time and clock vales are the first two 64-bit values in the clock
22     if(in->offset < <K>sizeof(UINT64) + sizeof(UINT64))
23     {
24         // Using Clock or Time so see if clock is running. Clock doesn't run while
25         // NV is unavailable.
26         // TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned here.
27         result = NvIsAvailable();
28         if(result != TPM_RC_SUCCESS)
29             return result;
30     }
31     // Get pointer to the session structure
32     session = SessionGet(in->policySession);
33
34     //If this is a trial policy, skip all validations and the operation
35     if(session->attributes.isTrialPolicy == CLEAR)
36     {
37         // Get time data info. The size of time info data equals the input
38         // operand B size. A TPM_RC_RANGE error may be returned at this point
39         result = TimeGetRange(in->offset, in->operandB.t.size, infoData);
40         if(result != TPM_RC_SUCCESS) return result;
41
42         // Arithmetic Comparison
43         switch(in->operation)
44         {
45             case TPM_EO_EQ:
46                 // compare A = B
47                 if(CryptCompare(in->operandB.t.size, infoData,
48                                 in->operandB.t.size, in->operandB.t.buffer) != 0)
49                     return TPM_RC_POLICY;
50                 break;
51             case TPM_EO_NEQ:
52                 // compare A != B

```

```

53         if(CryptCompare(in->operandB.t.size, infoData,
54             in->operandB.t.size, in->operandB.t.buffer) == 0)
55             return TPM_RC_POLICY;
56         break;
57     case TPM_EO_SIGNED_GT:
58         // compare A > B signed
59         if(CryptCompareSigned(in->operandB.t.size, infoData,
60             in->operandB.t.size, in->operandB.t.buffer) <= 0)
61             return TPM_RC_POLICY;
62         break;
63     case TPM_EO_UNSIGNED_GT:
64         // compare A > B unsigned
65         if(CryptCompare(in->operandB.t.size, infoData,
66             in->operandB.t.size, in->operandB.t.buffer) <= 0)
67             return TPM_RC_POLICY;
68         break;
69     case TPM_EO_SIGNED_LT:
70         // compare A < B signed
71         if(CryptCompareSigned(in->operandB.t.size, infoData,
72             in->operandB.t.size, in->operandB.t.buffer) >= 0)
73             return TPM_RC_POLICY;
74         break;
75     case TPM_EO_UNSIGNED_LT:
76         // compare A < B unsigned
77         if(CryptCompare(in->operandB.t.size, infoData,
78             in->operandB.t.size, in->operandB.t.buffer) >= 0)
79             return TPM_RC_POLICY;
80         break;
81     case TPM_EO_SIGNED_GE:
82         // compare A >= B signed
83         if(CryptCompareSigned(in->operandB.t.size, infoData,
84             in->operandB.t.size, in->operandB.t.buffer) < 0)
85             return TPM_RC_POLICY;
86         break;
87     case TPM_EO_UNSIGNED_GE:
88         // compare A >= B unsigned
89         if(CryptCompare(in->operandB.t.size, infoData,
90             in->operandB.t.size, in->operandB.t.buffer) < 0)
91             return TPM_RC_POLICY;
92         break;
93     case TPM_EO_SIGNED_LE:
94         // compare A <= B signed
95         if(CryptCompareSigned(in->operandB.t.size, infoData,
96             in->operandB.t.size, in->operandB.t.buffer) > 0)
97             return TPM_RC_POLICY;
98         break;
99     case TPM_EO_UNSIGNED_LE:
100         // compare A <= B unsigned
101         if(CryptCompare(in->operandB.t.size, infoData,
102             in->operandB.t.size, in->operandB.t.buffer) > 0)
103             return TPM_RC_POLICY;
104         break;
105     case TPM_EO_BITSET:
106         // All bits SET in B are SET in A. ((A&B)=B)
107     {
108         UINT32 i;
109         for (i = 0; i < in->operandB.t.size; i++)
110             if( (infoData[i] & in->operandB.t.buffer[i])
111                 != in->operandB.t.buffer[i])
112                 return TPM_RC_POLICY;
113     }
114     break;
115     case TPM_EO_BITCLEAR:
116         // All bits SET in B are CLEAR in A. ((A&B)=0)

```



```
117     {
118         UINT32 i;
119         for (i = 0; i < in->operandB.t.size; i++)
120             if((infoData[i] & in->operandB.t.buffer[i]) != 0)
121                 return TPM_RC_POLICY;
122     }
123     break;
124     default:
125         pAssert(FALSE);
126         break;
127 }
128 }
129
130 // Internal Data Update
131
132 // Start argument list hash
133 argHash.t.size = CryptStartHash(session->authHashAlg, &hashState);
134 // add operandB
135 CryptUpdateDigest2B(&hashState, &in->operandB.b);
136 // add offset
137 CryptUpdateDigestInt(&hashState, sizeof(UINT16), &in->offset);
138 // add operation
139 CryptUpdateDigestInt(&hashState, sizeof(TPM_EO), &in->operation);
140 // complete argument hash
141 CryptCompleteHash2B(&hashState, &argHash.b);
142
143 // update policyDigest
144 // start hash
145 CryptStartHash(session->authHashAlg, &hashState);
146
147 // add old digest, which may be empty
148 CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
149
150 // add commandCode
151 CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
152
153 // add argument digest
154 CryptUpdateDigest2B(&hashState, &argHash.b);
155
156 // complete the digest
157 CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
158
159 return TPM_RC_SUCCESS;
160 }
```

## 25.11 TPM2\_PolicyCommandCode

### 25.11.1 General Description

This command indicates that the authorization will be limited to a specific command code.

If *policySession*→*commandCode* has its default value, then it will be set to *code*. If *policySession*→*commandCode* does not have its default value, then the TPM will return TPM\_RC\_VALUE if the two values are not the same.

If *code* is not implemented, the TPM will return TPM\_RC\_POLICY\_CC.

If the TPM does not return an error, it will update *policySession*→*policyDigest* by

$$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM\_CC\_PolicyCommandCode || code) \quad (26)$$

NOTE 1 If a previous TPM2\_PolicyCommandCode() had been executed, then it is probable that the policy expression is improperly formed but the TPM does not return an error.

NOTE 2 A TPM2\_PolicyOR() would be used to allow an authorization to be used for multiple commands.

When the policy session is used to authorize a command, the TPM will fail the command if the *commandCode* of that command does not match *policySession*→*commandCode*.

This command, or TPM2\_PolicyDuplicationSelect(), is required to enable the policy to be used for ADMIN role authorization.

EXAMPLE Before TPM2\_Certify() can be executed, TPM2\_PolicyCommandCode() with *code* set to TPM\_CC\_Certify is required.

## 25.11.2 Command and Response

Table 129 — TPM2\_PolicyCommandCode Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyCommandCode
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None
TPM_CC	code	the allowed <i>commandCode</i>

Table 130 — TPM2\_PolicyCommandCode Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 25.11.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PolicyCommandCode_fp.h"

```

Error Returns	Meaning
TPM_RC_VALUE	<i>commandCode</i> of <i>policySession</i> previously set to a different value

```

3  TPM_RC
4  TPM2_PolicyCommandCode(
5      PolicyCommandCode_In *in          // IN: input parameter list
6  )
7  {
8      SESSION      *session;
9      TPM_CC       commandCode = TPM_CC_PolicyCommandCode;
10     HASH_STATE   hashState;
11
12     // Input validation
13
14     // Get pointer to the session structure
15     session = SessionGet(in->policySession);
16
17     if(session->commandCode != 0 && session->commandCode != in->code)
18         return TPM_RC_VALUE + RC_PolicyCommandCode_code;
19     if(!CommandIsImplemented(in->code))
20         return TPM_RC_POLICY_CC + RC_PolicyCommandCode_code;
21
22     // Internal Data Update
23     // Update policy hash
24     // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyCommandCode || code)
25     // Start hash
26     CryptStartHash(session->authHashAlg, &hashState);
27
28     // add old digest, which may be empty
29     CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
30
31     // add commandCode
32     CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
33
34     // add input commandCode
35     CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &in->code);
36
37     // complete the hash and get the results
38     CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
39
40     // update commandCode value in session context
41     session->commandCode = in->code;
42
43     return TPM_RC_SUCCESS;
44 }

```

## 25.12 TPM2\_PolicyPhysicalPresence

### 25.12.1 General Description

This command indicates that physical presence will need to be asserted at the time the authorization is performed.

If this command is successful, *policySession*→*isPPRequired* will be SET to indicate that this check is required when the policy is used for authorization. Additionally, *policySession*→*policyDigest* is extended with

$$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM\_CC\_PolicyPhysicalPresence) \quad (27)$$

## 25.12.2 Command and Response

Table 131 — TPM2\_PolicyPhysicalPresence Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyPhysicalPresence
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None

Table 132 — TPM2\_PolicyPhysicalPresence Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 25.12.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "PolicyPhysicalPresence_fp.h"
3  TPM_RC
4  TPM2_PolicyPhysicalPresence(
5      PolicyPhysicalPresence_In *in          // IN: input parameter list
6  )
7  {
8      SESSION      *session;
9      TPM_CC       commandCode = TPM_CC_PolicyPhysicalPresence;
10     HASH_STATE   hashState;
11
12     // Internal Data Update
13
14     // Get pointer to the session structure
15     session = SessionGet(in->policySession);
16
17     // Update policy hash
18     // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyPhysicalPresence)
19     // Start hash
20     CryptStartHash(session->authHashAlg, &hashState);
21
22     // add old digest, which may be empty
23     CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
24
25     // add commandCode
26     CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
27
28     // complete the digest
29     CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
30
31     // update session attribute
32     session->attributes.isPPRequired = SET;
33
34     return TPM_RC_SUCCESS;
35 }
```

## 25.13 TPM2\_PolicyCpHash

### 25.13.1 General Description

This command is used to allow a policy to be bound to a specific command and command parameters.

TPM2\_PolicySigned(), TPM2\_PolicySecret(), and TPM2\_PolicyTicket() are designed to allow an authorizing entity to execute an arbitrary command as the *cpHashA* parameter of those commands is not included in *policySession*→*policyDigest*. TPM2\_PolicyCommandCode() allows the policy to be bound to a specific Command Code so that only certain entities may authorize specific command codes. This command allows the policy to be restricted such that an entity may only authorize a command with a specific set of parameters.

If *policySession*→*cpHash* is already set and not the same as *cpHashA*, then the TPM shall return TPM\_RC\_VALUE. If *cpHashA* does not have the size of the *policySession*→*policyDigest*, the TPM shall return TPM\_RC\_SIZE.

If the *cpHashA* checks succeed, *policySession*→*cpHash* is set to *cpHashA* and *policySession*→*policyDigest* is updated with

$$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM\_CC\_PolicyCpHash || cpHashA) \quad (28)$$



## 25.13.2 Command and Response

Table 133 — TPM2\_PolicyCpHash Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyCpHash
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None
TPM2B_DIGEST	cpHashA	the <i>cpHash</i> added to the policy

Table 134 — TPM2\_PolicyCpHash Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 25.13.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PolicyCpHash_fp.h"

```

Error Returns	Meaning
TPM_RC_CPHASH	<i>cpHash</i> of <i>policySession</i> has previously been set to a different value
TPM_RC_SIZE	<i>cpHashA</i> is not the size of a digest produced by the hash algorithm associated with <i>policySession</i>

```

3  TPM_RC
4  TPM2_PolicyCpHash(
5      PolicyCpHash_In *in          // IN: input parameter list
6  )
7  {
8      SESSION      *session;
9      TPM_CC       commandCode = TPM_CC_PolicyCpHash;
10     HASH_STATE   hashState;
11
12     // Input Validation
13
14     // Get pointer to the session structure
15     session = SessionGet(in->policySession);
16
17     // A new cpHash is given in input parameter, but cpHash in session context
18     // is not empty, or is not the same as the new cpHash
19     if(   in->cpHashA.t.size != 0
20         && session->u1.cpHash.t.size != 0
21         && !Memory2BEqual(&in->cpHashA.b, &session->u1.cpHash.b)
22     )
23         return TPM_RC_CPHASH;
24
25     // A valid cpHash must have the same size as session hash digest
26     if(in->cpHashA.t.size != CryptGetHashDigestSize(session->authHashAlg))
27         return TPM_RC_SIZE + RC_PolicyCpHash_cpHashA;
28
29     // Internal Data Update
30
31     // Update policy hash
32     // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyCpHash || cpHashA)
33     // Start hash
34     CryptStartHash(session->authHashAlg, &hashState);
35
36     // add old digest, which may be empty
37     CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
38
39     // add commandCode
40     CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
41
42     // add cpHashA
43     CryptUpdateDigest2B(&hashState, &in->cpHashA.b);
44
45     // complete the digest and get the results
46     CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
47
48     // update cpHash in session context
49     session->u1.cpHash = in->cpHashA;
50     session->attributes.iscpHashDefined = SET;
51
52     return TPM_RC_SUCCESS;

```

53 }

## 25.14 TPM2\_PolicyNameHash

### 25.14.1 General Description

This command allows a policy to be bound to a specific set of handles without being bound to the parameters of the command. This is most useful for commands such as TPM2\_Duplicate() and for TPM2\_PCR\_Event() when the referenced PCR requires a policy.

The *nameHash* parameter should contain the digest of the Names associated with the handles to be used in the authorized command.

EXAMPLE For the TPM2\_Duplicate() command, two handles are provided. One is the handle of the object being duplicated and the other is the handle of the new parent. For that command, *nameHash* would contain:

$$nameHash := H_{policyAlg}(objectHandle \rightarrow Name || newParentHandle \rightarrow Name)$$

If *policySession*→*cpHash* is already set, the TPM shall return TPM\_RC\_VALUE. If the size of *nameHash* is not the size of *policySession*→*policyDigest*, the TPM shall return TPM\_RC\_SIZE. Otherwise, *policySession*→*cpHash* is set to *nameHash*.

If this command completes successfully, the *cpHash* of the authorized command will not be used for validation. Only the digest of the Names associated with the handles in the command will be used.

NOTE 1 This allows the space normally used to hold *policySession*→*cpHash* to be used for *policySession*→*nameHash* instead.

The *policySession*→*policyDigest* will be updated with

$$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM\_CC\_PolicyNameHash || nameHash) \quad (29)$$

NOTE 2 This command will often be used with TPM2\_PolicyAuthorize() where the owner of the object being duplicated provides approval for their object to be migrated to a specific new parent.

25.14.2 Command and Response

**Table 135 — TPM2\_PolicyNameHash Command**

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyNameHash
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None
TPM2B_DIGEST	nameHash	the digest to be added to the policy

**Table 136 — TPM2\_PolicyNameHash Response**

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 25.14.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PolicyNameHash_fp.h"

```

Error Returns	Meaning
TPM_RC_CPHASH	<i>nameHash</i> has been previously set to a different value
TPM_RC_SIZE	<i>nameHash</i> is not the size of the digest produced by the hash algorithm associated with <i>policySession</i>

```

3  TPM_RC
4  TPM2_PolicyNameHash(
5      PolicyNameHash_In  *in      // IN: input parameter list
6  )
7  {
8      SESSION              *session;
9      TPM_CC               commandCode = TPM_CC_PolicyNameHash;
10     HASH_STATE           hashState;
11
12     // Input Validation
13
14     // Get pointer to the session structure
15     session = SessionGet(in->policySession);
16
17     // A new nameHash is given in input parameter, but cpHash in session context
18     // is not empty
19     if(in->nameHash.t.size != 0 && session->ul.cpHash.t.size != 0)
20         return TPM_RC_CPHASH;
21
22     // A valid nameHash must have the same size as session hash digest
23     if(in->nameHash.t.size != CryptGetHashDigestSize(session->authHashAlg))
24         return TPM_RC_SIZE + RC_PolicyNameHash_nameHash;
25
26     // Internal Data Update
27
28     // Update policy hash
29     // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyNameHash || nameHash)
30     // Start hash
31     CryptStartHash(session->authHashAlg, &hashState);
32
33     // add old digest, which may be empty
34     CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
35
36     // add commandCode
37     CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
38
39     // add nameHash
40     CryptUpdateDigest2B(&hashState, &in->nameHash.b);
41
42     // complete the digest
43     CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
44
45     // clear iscpHashDefined bit to indicate now this field contains a nameHash
46     session->attributes.iscpHashDefined = CLEAR;
47
48     // update nameHash in session context
49     session->ul.cpHash = in->nameHash;
50
51     return TPM_RC_SUCCESS;
52 }

```

## 25.15 TPM2\_PolicyDuplicationSelect

### 25.15.1 General Description

This command allows qualification of duplication to allow duplication to a selected new parent.

If this command not used in conjunction with TPM2\_PolicyAuthorize(), then only the new parent is selected.

EXAMPLE When an object is created when the list of allowed duplication targets is known, the policy would be created with *includeObject* CLEAR.

NOTE 1 Only the new parent may be selected because, without TPM2\_PolicyAuthorize(), the Name of the Object to be duplicated would need to be known at the time that Object's policy is created. However, since the Name of the Object includes its policy, the Name is not known.

If used in conjunction with TPM2\_PolicyAuthorize(), then the authorizer of the new policy has the option of selecting just the new parent or of selecting both the new parent and the duplication Object..

NOTE 2 If the authorizing entity for an TPM2\_PolicyAuthorize() only specifies the new parent, then that authorization may be applied to the duplication of any number of other Objects. If the authorizing entity specifies both a new parent and the duplicated Object, then the authorization only applies to that pairing of Object and new parent.

If either *policySession*→*cpHash* or *policySession*→*nameHash* has been previously set, the TPM shall return TPM\_RC\_CPHASH. Otherwise, *policySession*→*nameHah* will be set to:

$$nameHash := H_{policyAlg}(objectName || newParentName) \quad (30)$$

NOTE 3 It is allowed that *policySesion*→*nameHash* and *policySession*→*cpHash* to share the same memory space.

The *policySession*→*policyDigest* will be updated according to the setting of *includeObject*. If equal to YES, *policySession*→*policyDigest* is updated by:

$$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM\_CC\_PolicyDuplicationSelect || objectName || newParentName || includeObject) \quad (31)$$

If *includeObject* is NO, *policySession*→*policyDigest* is updated by:

$$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM\_CC\_PolicyDuplicationSelect || newParentName || includeObject) \quad (32)$$

NOTE 4 *PolicySession*→*CpHash* receives the digest of both Names so that the check performed in TPM2\_Duplicate() may be the same regardless of which Names are included in *policySession*→*policyDigest*. This means that, when TPM2\_PolicyDuplicationSelect() is executed, it is only valid for a specific pair of duplication object and new parent.

If the command succeeds, *commandCode* in the policy session context is set to TPM\_CC\_Duplicate.

NOTE 5 The normal use of this command is before a TPM2\_PolicyAuthorize(). An authorized entity would approve a *policyDigest* that allowed duplication to a specific new parent. The authorizing entity may want to limit the authorization so that the approval allows only a specific object to be duplicated to the new parent. In that case, the authorizing entity would approve the *policyDigest* of equation (31).

## 25.15.2 Command and Response

Table 137 — TPM2\_PolicyDuplicationSelect Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyDuplicationSelect
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None
TPM2B_NAME	objectName	the Name of the object to be duplicated
TPM2B_NAME	newParentName	the Name of the new parent
TPMI_YES_NO	includeObject	if YES, the <i>objectName</i> will be included in the value in <i>policySession</i> → <i>policyDigest</i>

Table 138 — TPM2\_PolicyDuplicationSelect Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	



## 25.15.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PolicyDuplicationSelect_fp.h"

```

Error Returns	Meaning
TPM_RC_COMMAND_CODE	<i>commandCode</i> of <i>policySession</i> ; is not empty
TPM_RC_CPHASH	<i>cpHash</i> of <i>policySession</i> is not empty

```

3  TPM_RC
4  TPM2_PolicyDuplicationSelect(
5      PolicyDuplicationSelect_In *in           // IN: input parameter list
6  )
7  {
8      SESSION      *session;
9      HASH_STATE   hashState;
10     TPM_CC       commandCode = TPM_CC_PolicyDuplicationSelect;
11
12     // Input Validation
13
14     // Get pointer to the session structure
15     session = SessionGet(in->policySession);
16
17     // cpHash in session context must be empty
18     if(session->u1.cpHash.t.size != 0)
19         return TPM_RC_CPHASH;
20
21     // commandCode in session context must be empty
22     if(session->commandCode != 0)
23         return TPM_RC_COMMAND_CODE;
24
25     // Internal Data Update
26
27     // Update name hash
28     session->u1.cpHash.t.size = CryptStartHash(session->authHashAlg, &hashState);
29
30     // add objectName
31     CryptUpdateDigest2B(&hashState, &in->objectName.b);
32
33     // add new parent name
34     CryptUpdateDigest2B(&hashState, &in->newParentName.b);
35
36     // complete hash
37     CryptCompleteHash2B(&hashState, &session->u1.cpHash.b);
38
39     // update policy hash
40     // Old policyDigest size should be the same as the new policyDigest size since
41     // they are using the same hash algorithm
42     session->u2.policyDigest.t.size
43         = CryptStartHash(session->authHashAlg, &hashState);
44
45     // add old policy
46     CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
47
48     // add command code
49     CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
50
51     // add objectName
52     if(in->includeObject == YES)
53         CryptUpdateDigest2B(&hashState, &in->objectName.b);

```

```
54
55     // add new parent name
56     CryptUpdateDigest2B(&hashState, &in->newParentName.b);
57
58     // add includeObject
59     CryptUpdateDigestInt(&hashState, sizeof(TPMI_YES_NO), &in->includeObject);
60
61     // complete digest
62     CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
63
64     // clear iscpHashDefined bit to indicate now this field contains a nameHash
65     session->attributes.iscpHashDefined = CLEAR;
66
67     // set commandCode in session context
68     session->commandCode = TPM_CC_Duplicate;
69
70     return TPM_RC_SUCCESS;
71 }
```

## 25.16 TPM2\_PolicyAuthorize

### 25.16.1 General Description

This command allows policies to change. If a policy were static, then it would be difficult to add users to a policy. This command lets a policy authority sign a new policy so that it may be used in an existing policy.

The authorizing entity signs a structure that contains

$$aHash := H_{aHashAlg}(approvedPolicy || policyRef) \quad (33)$$

The *aHashAlg* is required to be the *nameAlg* of the key used to sign the *aHash*. The *aHash* value is then signed (symmetric or asymmetric) by *keySign*. That signature is then checked by the TPM in `TPM2_VerifySignature()` which produces a ticket by

$$HMAC(proof, (TPM\_ST\_VERIFIED || aHash || keySign \rightarrow Name)) \quad (34)$$

NOTE The reason for the validation is because of the expectation that the policy will be used multiple times and it is more efficient to check a ticket than to load an object each time to check a signature.

The ticket is then used in `TPM2_PolicyAuthorize()` to validate the parameters.

The *keySign* parameter is required to be a valid object name using *nameAlg* other than `TPM_ALG_NULL`. If the first two octets of *keySign* are not a valid hash algorithm, the TPM shall return `TPM_RC_HASH`. If the remainder of the Name is not the size of the indicated digest, the TPM shall return `TPM_RC_SIZE`.

The TPM validates that the *approvedPolicy* matches the current value of *policySession*→*policyDigest* and if not, shall return `TPM_RC_VALUE`.

The TPM then validates that the parameters to `TPM2_PolicyAuthorize()` match the values used to generate the ticket. If so, the TPM will reset *policySession*→*policyDigest* to a Zero Digest. Then it will create a `TPM2B_NAME` (*keyName*) using *keySign* and update *policySession*→*policyDigest* with `PolicyUpdate()` (see 25.2.3).

$$\text{PolicyUpdate}(TPM\_CC\_PolicyAuthorize, keyName, policyRef) \quad (35)$$

If the ticket is not valid, the TPM shall return `TPM_RC_POLICY`.

If *policySession* is a trial session, *policySession*→*policyDigest* is extended as if the ticket is valid without actual verification.

NOTE The unmarshaling process requires that a proper `TPMT_TK_VERIFIED` be provided for *checkTicket* but it may be a NULL Ticket.

## 25.16.2 Command and Response

Table 139 — TPM2\_PolicyAuthorize Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyAuthorize
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None
TPM2B_DIGEST	approvedPolicy	digest of the policy being approved
TPM2B_NONCE	policyRef	a policy qualifier
TPM2B_NAME	keySign	Name of a key that can sign a policy addition
TPMT_TK_VERIFIED	checkTicket	ticket validating that <i>approvedPolicy</i> and <i>policyRef</i> were signed by <i>keySign</i>

Table 140 — TPM2\_PolicyAuthorize Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 25.16.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PolicyAuthorize_fp.h"
3  #include "Policy_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_HASH	hash algorithm in <i>keyName</i> is not supported .
TPM_RC_SIZE	<i>keyName</i> is not the correct size for its hash algorithm
TPM_RC_VALUE	the current <i>policyDigest</i> of <i>policySession</i> does not match <i>approvedPolicy</i> , or <i>checkTicket</i> doesn't match the provided values

```

4  TPM_RC
5  TPM2_PolicyAuthorize(
6      PolicyAuthorize_In      *in      // IN: input parameter list
7  )
8  {
9      SESSION                  *session;
10     TPM2B_DIGEST              authHash;
11     HASH_STATE                hashState;
12     TPMT_TK_VERIFIED          ticket;
13     TPM_ALG_ID                hashAlg;
14     UINT16                    digestSize;
15
16     // Input Validation
17
18     // Get pointer to the session structure
19     session = SessionGet(in->policySession);
20
21     // Extract from the Name of the key, the algorithm used to compute it's Name
22     hashAlg = BYTE_ARRAY_TO_UINT16(in->keySign.t.name);
23
24     // 'keySign' parameter needs to use a supported hash algorithm, otherwise
25     // can't tell how large the digest should be
26     digestSize = CryptGetHashDigestSize(hashAlg);
27     if(digestSize == 0)
28         return TPM_RC_HASH + RC_PolicyAuthorize_keySign;
29
30     if(digestSize != (in->keySign.t.size - 2))
31         return TPM_RC_SIZE + RC_PolicyAuthorize_keySign;
32
33     //If this is a trial policy, skip all validations
34     if(session->attributes.isTrialPolicy == CLEAR)
35     {
36         // Check that "approvedPolicy" matches the current value of the
37         // policyDigest in policy session
38         if(!Memory2BEqual(&session->u2.policyDigest.b,
39                          &in->approvedPolicy.b))
40             return TPM_RC_VALUE + RC_PolicyAuthorize_approvedPolicy;
41
42         // Validate ticket TPMT_TK_VERIFIED
43         // Compute aHash. The authorizing object sign a digest
44         // aHash := hash(approvedPolicy || policyRef).
45         // Start hash
46         authHash.t.size = CryptStartHash(hashAlg, &hashState);
47
48         // add approvedPolicy
49         CryptUpdateDigest2B(&hashState, &in->approvedPolicy.b);
50

```

```
51     // add policyRef
52     CryptUpdateDigest2B(&hashState, &in->policyRef.b);
53
54     // complete hash
55     CryptCompleteHash2B(&hashState, &authHash.b);
56
57     // re-compute TPMT_TK_VERIFIED
58     TicketComputeVerified(in->checkTicket.hierarchy, &authHash,
59                          &in->keySign, &ticket);
60
61     // Compare ticket digest. If not match, return error
62     if(!Memory2BEqual(&in->checkTicket.digest.b, &ticket.digest.b))
63         return TPM_RC_VALUE+ RC_PolicyAuthorize_checkTicket;
64 }
65
66 // Internal Data Update
67
68 // Set policyDigest to zero digest
69 MemorySet(session->u2.policyDigest.t.buffer, 0, session->u2.policyDigest.t.size);
70
71 // Update policyDigest
72 PolicyUpdate(TPM_CC_PolicyAuthorize, &in->keySign, &in->policyRef, session);
73
74 return TPM_RC_SUCCESS;
75
76 }
```

## 25.17 TPM2\_PolicyAuthValue

### 25.17.1 General Description

This command allows a policy to be bound to the authorization value of the authorized object.

When this command completes successfully, *policySession*→*isAuthValueNeeded* is SET to indicate that the *authValue* will be included in *hmacKey* when the authorization HMAC is computed for this session. Additionally, *policySession*→*isPasswordNeeded* will be CLEAR.

NOTE If a policy does not use this command, then the *hmacKey* for the authorized command would only use *sessionKey*. If *sessionKey* is not present, then the *hmacKey* is an Empty Buffer and no HMAC would be computed.

If successful, *policySession*→*policyDigest* will be updated with

$$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM\_CC\_PolicyAuthValue) \quad (36)$$

## 25.17.2 Command and Response

Table 141 — TPM2\_PolicyAuthValue Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyAuthValue
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None

Table 142 — TPM2\_PolicyAuthValue Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	



## 25.17.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "PolicyAuthValue_fp.h"
3  #include "Policy_spt_fp.h"
4  TPM_RC
5  TPM2_PolicyAuthValue(
6      PolicyAuthValue_In *in          // IN: input parameter list
7  )
8  {
9      SESSION          *session;
10     TPM_CC            commandCode = TPM_CC_PolicyAuthValue;
11     HASH_STATE        hashState;
12
13     // Internal Data Update
14
15     // Get pointer to the session structure
16     session = SessionGet(in->policySession);
17
18     // Update policy hash
19     // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyAuthValue)
20     // Start hash
21     CryptStartHash(session->authHashAlg, &hashState);
22
23     // add old digest, which may be empty
24     CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
25
26     // add commandCode
27     CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
28
29     // complete the hash and get the results
30     CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
31
32     // update isAuthValueNeeded bit in the session context
33     session->attributes.isAuthValueNeeded = SET;
34     session->attributes.isPasswordNeeded = CLEAR;
35
36     return TPM_RC_SUCCESS;
37 }
```

## 25.18 TPM2\_PolicyPassword

### 25.18.1 General Description

This command allows a policy to be bound to the authorization value of the authorized object.

When this command completes successfully, *policySession*→*isPasswordNeeded* is SET to indicate that *authValue* of the authorized object will be checked when the session is used for authorization. The caller will provide the *authValue* in clear text in the *hmac* parameter of the authorization. The comparison of *hmac* to *authValue* is performed as if the authorization is a password.

NOTE 1           The parameter field in the policy session where the authorization value is provided is called *hmac*. If TPM2\_PolicyPassword() is part of the sequence, then the field will contain a password and not an HMAC.

If successful, *policySession*→*policyDigest* will be updated with

$$policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM\_CC\_PolicyAuthValue) \quad (37)$$

NOTE 2           This is the same extend value as used with TPM2\_PolicyAuthValue so that the evaluation may be done using either an HMAC or a password with no change to the *authPolicy* of the object. The reason that two commands are present is to indicate to the TPM if the *hmac* field in the authorization will contain an HMAC or a password value.

When this command is successful, *policySession*→*isAuthValueNeeded* will be CLEAR.

25.18.2 Command and Response

Table 143 — TPM2\_PolicyPassword Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyPassword
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None

Table 144 — TPM2\_PolicyPassword Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 25.18.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "PolicyPassword_fp.h"
3  #include "Policy_spt_fp.h"
4  TPM_RC
5  TPM2_PolicyPassword(
6      PolicyPassword_In  *in      // IN: input parameter list
7  )
8  {
9      SESSION            *session;
10     TPM_CC              commandCode = TPM_CC_PolicyAuthValue;
11     HASH_STATE          hashState;
12
13     // Internal Data Update
14
15     // Get pointer to the session structure
16     session = SessionGet(in->policySession);
17
18     // Update policy hash
19     // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyAuthValue)
20     // Start hash
21     CryptStartHash(session->authHashAlg, &hashState);
22
23     // add old digest, which may be empty
24     CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
25
26     // add commandCode
27     CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
28
29     // complete the digest
30     CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
31
32     // Update isPasswordNeeded bit
33     session->attributes.isPasswordNeeded = SET;
34     session->attributes.isAuthValueNeeded = CLEAR;
35
36     return TPM_RC_SUCCESS;
37 }
```

## 25.19 TPM2\_PolicyGetDigest

### 25.19.1 General Description

This command returns the current *policyDigest* of the session. This command allows the TPM to be used to perform the actions required to pre-compute the *authPolicy* for an object.

## 25.19.2 Command and Response

Table 145 — TPM2\_PolicyGetDigest Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyGetDigest
TPMI_SH_POLICY	policySession	handle for the policy session Auth Index: None

Table 146 — TPM2\_PolicyGetDigest Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DIGEST	policyDigest	the current value of the <i>policySession</i> → <i>policyDigest</i>

### 25.19.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "PolicyGetDigest_fp.h"
3  TPM_RC
4  TPM2_PolicyGetDigest(
5      PolicyGetDigest_In      *in,           // IN: input parameter list
6      PolicyGetDigest_Out     *out          // OUT: output parameter list
7  )
8  {
9      SESSION      *session;
10
11  // Command Output
12
13  // Get pointer to the session structure
14  session = SessionGet(in->policySession);
15
16  out->policyDigest = session->u2.policyDigest;
17
18  return TPM_RC_SUCCESS;
19 }
```

## 26 Hierarchy Commands

### 26.1 TPM2\_CreatePrimary

#### 26.1.1 General Description

This command is used to create a Primary Object under one of the Primary Seeds or a Temporary Object under TPM\_RH\_NULL. The command uses a TPM2B\_PUBLIC as a template for the object to be created. The command will create and load a Primary Object. The sensitive area is not returned.

Any type of object and attributes combination that is allowed by TPM2\_Create() may be created by this command. The constraints on templates and parameters are the same as TPM2\_Create() except that a Primary Storage Key and a Temporary Storage Key are not constrained to use the algorithms of their parents.

For setting of the attributes of the created object, *fixedParent*, *fixedTPM*, *userWithAuth*, *adminWithPolicy*, *encrypt*, and *restricted* are implied to be SET in the parent (a Permanent Handle). The remaining attributes are implied to be CLEAR.

The TPM will derive the object from the Primary Seed indicated in *primaryHandle* using an approved KDF. All of the bits of the template are used in the creation of the Primary Key. Methods for creating a Primary Object from a Primary Seed are described in Part 1 of this specification and implemented in Part 4.

If this command is called multiple times with the same *inPublic* parameter, *inSensitive.data*, and Primary Seed, the TPM shall produce the same Primary Object.

NOTE            If the Primary Seed is changed, the Primary Objects generated with the new seed shall be statistically unique even if the parameters of the call are the same.

This command requires authorization. Authorization for a Primary Object attached to the Platform Primary Seed (PPS) shall be provided by *platformAuth* or *platformPolicy*. Authorization for a Primary Object attached to the Storage Primary Seed (SPS) shall be provided by *ownerAuth* or *ownerPolicy*. Authorization for a Primary Key attached to the Endorsement Primary Seed (EPS) shall be provided by *endorsementAuth* or *endorsementPolicy*.



26.1.2 Command and Response

Table 147 — TPM2\_CreatePrimary Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_CreatePrimary
TPMI_RH_HIERARCHY+	@primaryHandle	TPM_RH_ENDORSEMENT, TPM_RH_OWNER, TPM_RH_PLATFORM+{PP}, or TPM_RH_NULL Auth Index: 1 Auth Role: USER
TPM2B_SENSITIVE_CREATE	inSensitive	the sensitive data
TPM2B_PUBLIC	inPublic	the public template
TPM2B_DATA	outsideInfo	data that will be included in the creation data for this object to provide permanent, verifiable linkage between this object and some object owner data
TPML_PCR_SELECTION	creationPCR	PCR that will be used in creation data

Table 148 — TPM2\_CreatePrimary Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM_HANDLE	objectHandle	Handle for created Primary Object
TPM2B_PUBLIC	outPublic	the public portion of the created object
TPM2B_CREATION_DATA	creationData	contains a TPMT_CREATION_DATA
TPM2B_DIGEST	creationHash	digest of <i>creationData</i> using <i>nameAlg</i> of <i>outPublic</i>
TPMT_TK_CREATION	creationTicket	ticket used by TPM2_CertifyCreation() to validate that the creation data was produced by the TPM
TPM2B_NAME	name	the name of the created object

## 26.1.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "CreatePrimary_fp.h"
3  #include "Object_spt_fp.h"
4  #include <Platform.h>

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	<i>sensitiveDataOrigin</i> is CLEAR when 'sensitive. data' is an Empty Buffer, or is SET when 'sensitive. data' is not empty; <i>fixedTPM</i> , <i>fixedParent</i> , or <i>encryptedDuplication</i> attributes are inconsistent between themselves or with those of the parent object; inconsistent <i>restricted</i> , <i>decrypt</i> and <i>sign</i> attributes; attempt to inject sensitive data for an asymmetric key; attempt to create a symmetric cipher key that is not a decryption key
TPM_RC_KDF	incorrect KDF specified for decrypting keyed hash object
TPM_RC_OBJECT_MEMORY	there is no free slot for the object
TPM_RC_SCHEME	inconsistent attributes <i>decrypt</i> , <i>sign</i> , <i>restricted</i> and key's scheme ID; or hash algorithm is inconsistent with the scheme ID for keyed hash object
TPM_RC_SIZE	size of public auth policy or sensitive auth value does not match digest size of the name algorithm sensitive data size for the keyed hash object is larger than is allowed for the scheme
TPM_RC_SYMMETRIC	a storage key with no symmetric algorithm specified; or non-storage key with symmetric algorithm different from TPM_ALG_NULL
TPM_RC_TYPE	unknown object type;

```

5  TPM_RC
6  TPM2_CreatePrimary(
7      CreatePrimary_In  *in,          // IN: input parameter list
8      CreatePrimary_Out *out         // OUT: output parameter list
9  )
10 {
11     // Local variables
12     TPM_RC          result = TPM_RC_SUCCESS;
13     TPMT_SENSITIVE sensitive;
14
15     // Input Validation
16     // The sensitiveDataOrigin attribute must be consistent with the setting of
17     // the size of the data object in inSensitive.
18     if( (in->inPublic.t.publicArea.objectAttributes.sensitiveDataOrigin == SET)
19         != (in->inSensitive.t.sensitive.data.t.size == 0 ) )
20         // Mismatch between the object attributes and the parameter.
21         return TPM_RC_ATTRIBUTES + RC_CreatePrimary_inSensitive;
22
23     // Check attributes in input public area. TPM_RC_ATTRIBUTES, TPM_RC_KDF,
24     // TPM_RC_SCHEME, TPM_RC_SIZE, TPM_RC_SYMMETRIC, or TPM_RC_TYPE error may
25     // be returned at this point.
26     result = PublicAttributesValidation(FALSE, in->primaryHandle,
27                                         &in->inPublic.t.publicArea);
28     if(result != TPM_RC_SUCCESS)
29         return RcSafeAddToResult(result, RC_CreatePrimary_inPublic);
30
31     // Validate the sensitive area values
32     if( MemoryRemoveTrailingZeros(&in->inSensitive.t.sensitive.userAuth)
33         > CryptGetHashDigestSize(in->inPublic.t.publicArea.nameAlg) )
34         return TPM_RC_SIZE + RC_CreatePrimary_inSensitive;
35

```

```
36 // Command output
37
38 // Generate Primary Object
39 // The primary key generation process uses the Name of the input public
40 // template to compute the key. The keys are generated from the template
41 // before anything in the template is allowed to be changed.
42 // A TPM_RC_KDF, TPM_RC_SIZE error may be returned at this point
43 result = CryptCreateObject(in->primaryHandle, &in->inPublic.t.publicArea,
44                          &in->inSensitive.t.sensitive, &sensitive);
45 if(result != TPM_RC_SUCCESS)
46     return result;
47
48 // Fill in creation data
49 FillInCreationData(in->primaryHandle, in->inPublic.t.publicArea.nameAlg,
50                  &in->creationPCR, &in->outsideInfo, &out->creationData,
51                  &out->creationHash);
52
53 // Copy public area
54 out->outPublic = in->inPublic;
55
56 // Fill in private area for output
57 ObjectComputeName(&(out->outPublic.t.publicArea), &out->name);
58
59 // Compute creation ticket
60 TicketComputeCreation(EntityGetHierarchy(in->primaryHandle), &out->name,
61                          &out->creationHash, &out->creationTicket);
62
63 // Create a internal object. A TPM_RC_OBJECT_MEMORY error may be returned
64 // at this point.
65 result = ObjectLoad(in->primaryHandle, &in->inPublic.t.publicArea, &sensitive,
66                  &out->name, in->primaryHandle, TRUE, &out->objectHandle);
67
68 return result;
69 }
```

## 26.2 TPM2\_HierarchyControl

### 26.2.1 General Description

This command enables and disables use of a hierarchy. The command allows *phEnable*, *shEnable*, and *ehEnable* to be changed when the proper authorization is provided.

This command may be used to CLEAR *phEnable* if *platformAuth/platformPolicy* is provided. *phEnable* may not be SET using this command.

This command may be used to CLEAR *shEnable* if either *platformAuth/platformPolicy* or *ownerAuth/ownerPolicy* is provided. *shEnable* may be SET if *platformAuth/platformPolicy* is provided.

This command may be used to CLEAR *ehEnable* if either *platformAuth/platformPolicy* or *endorsementAuth/endorsementPolicy* is provided. *ehEnable* may be SET if *platformAuth/platformPolicy* is provided.

When this command is used to CLEAR an enable, the TPM will disable use of any persistent entity associated with the disabled hierarchy and to flush any transient objects associated with the disabled hierarchy.

- a) If an NV Index has TPMA\_NV\_PLATFORMCREATE SET (indicating that the NV Index was defined using *platformAuth*) and *phEnable* is CLEAR:
  - 1) the NV Index may only be read if TPMA\_NV\_OWNERREAD is SET and the authorization handle is TPM\_RH\_OWNER; and
  - 2) the NV Index may only be updated if TPMA\_NV\_OWNERWRITE is SET and the authorization handle is TPM\_RH\_OWNER.
- b) If an NV Index has TPMA\_NV\_PLATFORMCREATE is CLEAR (indicating that the NV Index was defined using *ownerAuth*) and *shEnable* is CLEAR:
  - 1) the NV Index may only be read if TPMA\_NV\_PPREAD is SET and the authorization handle is TPM\_RH\_PLATFORM; and
  - 2) the NV Index may only be updated if TPMA\_NV\_PPWRITE is SET and the authorization handle is TPM\_RH\_PLATFORM.

•

26.2.2 Command and Response

Table 149 — TPM2\_HierarchyControl Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_HierarchyControl {NV E}
TPMI_RH_HIERARCHY	@authHandle	TPM_RH_ENDORSEMENT, TPM_RH_OWNER or TPM_RH_PLATFORM+{PP} Auth Index: 1 Auth Role: USER
TPMI_RH_HIERARCHY	hierarchy	hierarchy of the enable being modified TPM_RH_ENDORSEMENT, TPM_RH_OWNER or TPM_RH_PLATFORM
TPMI_YES_NO	state	YES if the enable should be SET, NO if the enable should be CLEAR

Table 150 — TPM2\_HierarchyControl Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 26.2.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "HierarchyControl_fp.h"

```

Error Returns	Meaning
TPM_RC_AUTH_TYPE	<i>authHandle</i> is not applicable to <i>hierarchy</i> in its current state

```

3  TPM_RC
4  TPM2_HierarchyControl(
5      HierarchyControl_In      *in          // IN: input parameter list
6  )
7  {
8      TPM_RC      result;
9
10     // This command may cause the orderlyState to be cleared due to
11     // the update of state clear data.  If this is the case, check if NV is
12     // available first
13     if(gp.orderlyState != SHUTDOWN_NONE)
14     {
15         // The command needs NV update.  Check if NV is available.
16         // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
17         // this point
18         result = NvIsAvailable();
19         if(result != TPM_RC_SUCCESS) return result;
20     }
21
22     // Input Validation
23     switch(in->hierarchy)
24     {
25         // Platform hierarchy has to be disabled by platform auth
26         // If the platform hierarchy has already been disabled, only a reboot
27         // can enable it again
28     case TPM_RH_PLATFORM:
29         if(in->authHandle != TPM_RH_PLATFORM)
30             return TPM_RC_AUTH_TYPE;
31         break;
32
33         // ShEnable may be disabled if PlatformAuth/PlatformPolicy or
34         // OwnerAuth/OwnerPolicy is provided.  If ShEnable is disabled, then it
35         // may only be enabled if PlatformAuth/PlatformPolicy is provided.
36     case TPM_RH_OWNER:
37         if( in->authHandle != TPM_RH_PLATFORM
38            && in->authHandle != TPM_RH_OWNER)
39             return TPM_RC_AUTH_TYPE;
40         if( gc.shEnable == FALSE && in->state == YES
41            && in->authHandle != TPM_RH_PLATFORM)
42             return TPM_RC_AUTH_TYPE;
43         break;
44
45         // EhEnable may be disabled if either PlatformAuth/PlatformPolicy or
46         // EndorsementAuth/EndorsementPolicy is provided.  If EhEnable is disabled,
47         // then it may only be enabled if PlatformAuth/PlatformPolicy is
48         // provided.
49     case TPM_RH_ENDORSEMENT:
50         if( in->authHandle != TPM_RH_PLATFORM
51            && in->authHandle != TPM_RH_ENDORSEMENT)
52             return TPM_RC_AUTH_TYPE;
53         if( gc.ehEnable == FALSE && in->state == YES
54            && in->authHandle != TPM_RH_PLATFORM)

```

```
55         return TPM_RC_AUTH_TYPE;
56         break;
57
58     default:
59         pAssert(FALSE);
60         break;
61     }
62
63 // Internal Data Update
64
65 // Enable or disable hierarchy
66 switch(in->hierarchy)
67 {
68     case TPM_RH_OWNER:
69         if(in->state == YES)
70             gc.shEnable = TRUE;
71         else
72             gc.shEnable = FALSE;
73         break;
74     case TPM_RH_ENDORSEMENT:
75         if(in->state == YES)
76             gc.ehEnable = TRUE;
77         else
78             gc.ehEnable = FALSE;
79         break;
80     case TPM_RH_PLATFORM:
81         if(in->state == YES)
82             g_phEnable = TRUE;
83         else
84             g_phEnable = FALSE;
85         break;
86     default:
87         pAssert(FALSE);
88         break;
89 }
90
91 if(in->state == NO)
92     // Flush hierarchy
93     ObjectFlushHierarchy(in->hierarchy);
94
95 // orderly state should be cleared because of the update to state clear data
96 g_clearOrderly = TRUE;
97
98 return TPM_RC_SUCCESS;
99 }
```

## 26.3 TPM2\_SetPrimaryPolicy

### 26.3.1 General Description

This command allows setting of the authorization policy for the platform hierarchy (*platformPolicy*), the storage hierarchy (*ownerPolicy*), and the endorsement hierarchy (*endorsementPolicy*).

The command requires an authorization session. The session shall use the current *authValue* or satisfy the current *authPolicy* for the referenced hierarchy.

The policy that is changed is the policy associated with *authHandle*.

If the enable associated with *authHandle* is not SET, then the associated authorization values (*authValue* or *authPolicy*) may not be used.



26.3.2 Command and Response

Table 151 — TPM2\_SetPrimaryPolicy Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_SetPrimaryPolicy {NV}
TPMI_RH_HIERARCHY	@authHandle	TPM_RH_ENDORSEMENT, TPM_RH_OWNER or TPM_RH_PLATFORM+{PP} Auth Index: 1 Auth Role: USER
TPM2B_DIGEST	authPolicy	an authorization policy digest; may be the Empty Buffer If <i>hashAlg</i> is TPM_ALG_NULL, then this shall be an Empty Buffer.
TPMI_ALG_HASH+	hashAlg	the hash algorithm to use for the policy If the <i>authPolicy</i> is an Empty Buffer, then this field shall be TPM_ALG_NULL.

Table 152 — TPM2\_SetPrimaryPolicy Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 26.3.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "SetPrimaryPolicy_fp.h"

```

Error Returns	Meaning
TPM_RC_SIZE	size of input <i>authPolicy</i> is not consistent with input hash algorithm

```

3  TPM_RC
4  TPM2_SetPrimaryPolicy(
5      SetPrimaryPolicy_In    *in        // IN: input parameter list
6  )
7  {
8      TPM_RC                result;
9
10     // Input Validation
11
12     // Check the authPolicy consistent with hash algorithm
13     if( in->authPolicy.t.size != 0
14         && in->authPolicy.t.size != CryptGetHashDigestSize(in->hashAlg))
15         return TPM_RC_SIZE + RC_SetPrimaryPolicy_authPolicy;
16
17     // The command need NV update for OWNER and ENDORSEMENT hierarchy, and
18     // might need orderlyState update for PLATFORM hierarchy.
19     // Check if NV is available. A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE
20     // error may be returned at this point
21     result = NvIsAvailable();
22     if(result != TPM_RC_SUCCESS)
23         return result;
24
25     // Internal Data Update
26
27     // Set hierarchy policy
28     switch(in->authHandle)
29     {
30         case TPM_RH_OWNER:
31             gp.ownerAlg = in->hashAlg;
32             gp.ownerPolicy = in->authPolicy;
33             NvWriteReserved(NV_OWNER_ALG, &gp.ownerAlg);
34             NvWriteReserved(NV_OWNER_POLICY, &gp.ownerPolicy);
35             break;
36         case TPM_RH_ENDORSEMENT:
37             gp.endorsementAlg = in->hashAlg;
38             gp.endorsementPolicy = in->authPolicy;
39             NvWriteReserved(NV_ENDORSEMENT_ALG, &gp.endorsementAlg);
40             NvWriteReserved(NV_ENDORSEMENT_POLICY, &gp.endorsementPolicy);
41             break;
42         case TPM_RH_PLATFORM:
43             gc.platformAlg = in->hashAlg;
44             gc.platformPolicy = in->authPolicy;
45             // need to update orderly state
46             g_clearOrderly = TRUE;
47             break;
48         default:
49             pAssert(FALSE);
50             break;
51     }
52
53     return TPM_RC_SUCCESS;
54 }

```

## 26.4 TPM2\_ChangePPS

### 26.4.1 General Description

This replaces the current PPS with a value from the RNG and sets *platformPolicy* to the default initialization value (the Empty Buffer).

NOTE 1            A policy that is the Empty Buffer can match no policy.

NOTE 2            *platformAuth* is not changed.

All loaded transient and persistent objects in the Platform hierarchy are flushed.

Saved contexts in the Platform hierarchy that were created under the old PPS will no longer be able to be loaded.

The policy hash algorithm for PCR is reset to TPM\_ALG\_NULL.

This command does not clear any NV Index values.

NOTE 3            Index values belonging to the Platform are preserved because the indexes may have configuration information that will be the same after the PPS changes. The Platform may remove the indexes that are no longer needed using TPM2\_NV\_UndefineSpace().

This command requires *platformAuth*.

## 26.4.2 Command and Response

Table 153 — TPM2\_ChangePPS Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ChangePPS {NV E}
TPMI_RH_PLATFORM	@authHandle	TPM_RH_PLATFORM+{PP} Auth Index: 1 Auth Role: USER

Table 154 — TPM2\_ChangePPS Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 26.4.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "ChangePPS_fp.h"
3  TPM_RC
4  TPM2_ChangePPS(
5      ChangePPS_In      *in          // IN: input parameter list
6  )
7  {
8      UINT32             i;
9      TPM_RC             result;
10
11     // Check if NV is available. A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE
12     // error may be returned at this point
13     result = NvIsAvailable();
14     if(result != TPM_RC_SUCCESS) return result;
15
16     // Input parameter is not reference in command action
17     in = NULL;
18
19     // Internal Data Update
20
21     // Reset platform hierarchy seed from RNG
22     CryptGenerateRandom(PRIMARY_SEED_SIZE, gp.PPSeed.t.buffer);
23
24     // Create a new phProof value from RNG to prevent the saved platform
25     // hierarchy contexts being loaded
26     CryptGenerateRandom(PROOF_SIZE, gp.phProof.t.buffer);
27
28     // Set platform authPolicy to null
29     gc.platformAlg = TPM_ALG_NULL;
30     gc.platformPolicy.t.size = 0;
31
32     // Flush loaded object in platform hierarchy
33     ObjectFlushHierarchy(TPM_RH_PLATFORM);
34
35     // Flush platform evict object and index in NV
36     NvFlushHierarchy(TPM_RH_PLATFORM);
37
38     // Save hierarchy changes to NV
39     NvWriteReserved(NV_PP_SEED, &gp.PPSeed);
40     NvWriteReserved(NV_PH_PROOF, &gp.phProof);
41
42     // Re-initialize PCR policies
43     for(i = 0; i < NUM_POLICY_PCR_GROUP; i++)
44     {
45         gp.pcrPolicies.hashAlg[i] = TPM_ALG_NULL;
46         gp.pcrPolicies.policy[i].t.size = 0;
47     }
48     NvWriteReserved(NV_PCR_POLICIES, &gp.pcrPolicies);
49
50     // orderly state should be cleared because of the update to state clear data
51     g_clearOrderly = TRUE;
52
53     return TPM_RC_SUCCESS;
54 }

```

## 26.5 TPM2\_ChangeEPS

### 26.5.1 General Description

This replaces the current EPS with a value from the RNG and sets the Endorsement hierarchy controls to their default initialization values: *ehEnable* is SET, *endorsementAuth* and *endorsementPolicy* both equal to the Empty Buffer. It will flush any loaded objects in the EPS hierarchy and not allow objects in the hierarchy associated with the previous EPS to be loaded.

NOTE            In the reference implementation, *ehProof* is a non-volatile value from the RNG. It is allowed that the *ehProof* be generated by a KDF using both the EPS and SPS as inputs. If generated with a KDF, the *ehProof* can be generated on an as-needed basis or made a non-volatile value.

This command requires *platformAuth*.

## 26.5.2 Command and Response

Table 155 — TPM2\_ChangeEPS Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ChangeEPS {NV E}
TPMI_RH_PLATFORM	@authHandle	TPM_RH_PLATFORM+{PP} Auth Handle: 1 Auth Role: USER

Table 156 — TPM2\_ChangeEPS Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 26.5.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "ChangeEPS_fp.h"
3  TPM_RC
4  TPM2_ChangeEPS(
5      ChangeEPS_In      *in          // IN: input parameter list
6  )
7  {
8      TPM_RC      result;
9
10     // The command needs NV update. Check if NV is available.
11     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
12     // this point
13     result = NvIsAvailable();
14     if(result != TPM_RC_SUCCESS) return result;
15
16     // Input parameter is not reference in command action
17     in = NULL;
18
19     // Internal Data Update
20
21     // Reset endorsement hierarchy seed from RNG
22     CryptGenerateRandom(PRIMARY_SEED_SIZE, gp.EPSeed.t.buffer);
23
24     // Create new ehProof value from RNG
25     CryptGenerateRandom(PROOF_SIZE, gp.ehProof.t.buffer);
26
27     // Enable endorsement hierarchy
28     gc.ehEnable = TRUE;
29
30     // set authValue buffer to zeros
31     MemorySet(gp.endorsementAuth.t.buffer, 0, gp.endorsementAuth.t.size);
32     // Set endorsement authValue to null
33     gp.endorsementAuth.t.size = 0;
34
35     // Set endorsement authPolicy to null
36     gp.endorsementAlg = TPM_ALG_NULL;
37     gp.endorsementPolicy.t.size = 0;
38
39     // Flush loaded object in endorsement hierarchy
40     ObjectFlushHierarchy(TPM_RH_ENDORSEMENT);
41
42     // Flush evict object of endorsement hierarchy stored in NV
43     NvFlushHierarchy(TPM_RH_ENDORSEMENT);
44
45     // Save hierarchy changes to NV
46     NvWriteReserved(NV_EP_SEED, &gp.EPSeed);
47     NvWriteReserved(NV_EH_PROOF, &gp.ehProof);
48     NvWriteReserved(NV_ENDORSEMENT_AUTH, &gp.endorsementAuth);
49     NvWriteReserved(NV_ENDORSEMENT_ALG, &gp.endorsementAlg);
50     NvWriteReserved(NV_ENDORSEMENT_POLICY, &gp.endorsementPolicy);
51
52     // orderly state should be cleared because of the update to state clear data
53     g_clearOrderly = TRUE;
54
55     return TPM_RC_SUCCESS;
56 }

```



## 26.6 TPM2\_Clear

### 26.6.1 General Description

This command removes all TPM context associated with a specific Owner.

The clear operation will:

- flush loaded objects (persistent and volatile) in the Storage and Endorsement hierarchies;
- delete any NV Index with TPMA\_NV\_PLATFORMCREATE == CLEAR;
- change the SPS to a new value from the TPM's random number generator (RNG),
- change shProof and ehProof,

**NOTE** The proof values may be set from the RNG or derived from the associated new Primary Seed. If derived from the Primary Seeds, the derivation of *ehProof* shall use both the SPS and EPS. The computation shall use the SPS as an HMAC key and the derived value may then be a parameter in a second HMAC in which the EPS is the HMAC key. The reference design uses values from the RNG.

- SET shEnable and ehEnable;
- set ownerAuth, endorsementAuth, and lockoutAuth to the Empty Buffer;
- set *ownerPolicy* and *endorsementPolicy* to the Empty Buffer;
- set *Clock* to zero;
- set *resetCount* to zero;
- set *restartCount* to zero; and
- set *Safe* to YES.

This command requires *platformAuth* or *lockoutAuth*. If TPM2\_ClearControl() has disabled this command, the TPM shall return TPM\_RC\_DISABLED.

If this command is authorized using *lockoutAuth*, the HMAC in the response shall use the new *lockoutAuth* value (that is, the Empty Buffer) when computing response HMAC.

## 26.6.2 Command and Response

Table 157 — TPM2\_Clear Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Clear {NV E}
TPMI_RH_CLEAR	@authHandle	TPM_RH_LOCKOUT or TPM_RH_PLATFORM+{PP} Auth Handle: 1 Auth Role: USER

Table 158 — TPM2\_Clear Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 26.6.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Clear_fp.h"

```

Error Returns	Meaning
TPM_RC_DISABLED	Clear command has been disabled

```

3  TPM_RC
4  TPM2_Clear(
5      Clear_In      *in          // IN: input parameter list
6  )
7  {
8      TPM_RC          result;
9
10     // Input parameter is not reference in command action
11     in = NULL;
12
13     // The command needs NV update. Check if NV is available.
14     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
15     // this point
16     result = NvIsAvailable();
17     if(result != TPM_RC_SUCCESS) return result;
18
19     // Input Validation
20
21     // If Clear command is disabled, return an error
22     if(gp.disableClear)
23         return TPM_RC_DISABLED;
24
25     // Internal Data Update
26
27     // Reset storage hierarchy seed from RNG
28     CryptGenerateRandom(PRIMARY_SEED_SIZE, gp.SPSeed.t.buffer);
29
30     // Create new shProof and ehProof value from RNG
31     CryptGenerateRandom(PROOF_SIZE, gp.shProof.t.buffer);
32     CryptGenerateRandom(PROOF_SIZE, gp.ehProof.t.buffer);
33
34     // Enable storage and endorsement hierarchy
35     gc.shEnable = gc.ehEnable = TRUE;
36
37     // set the authValue buffers to zero
38     MemorySet(gp.ownerAuth.t.buffer, 0, gp.ownerAuth.t.size);
39     MemorySet(gp.endorsementAuth.t.buffer, 0, gp.endorsementAuth.t.size);
40     MemorySet(gp.lockoutAuth.t.buffer, 0, gp.lockoutAuth.t.size);
41     // Set storage, endorsement and lockout authValue to null
42     gp.ownerAuth.t.size = gp.endorsementAuth.t.size = gp.lockoutAuth.t.size = 0;
43
44     // Set storage and endorsement authPolicy to null
45     gp.ownerAlg = gp.endorsementAlg = TPM_ALG_NULL;
46     gp.ownerPolicy.t.size = gp.endorsementPolicy.t.size = 0;
47
48     // Flush loaded object in storage and endorsement hierarchy
49     ObjectFlushHierarchy(TPM_RH_OWNER);
50     ObjectFlushHierarchy(TPM_RH_ENDORSEMENT);
51
52     // Flush owner and endorsement object and owner index in NV
53     NvFlushHierarchy(TPM_RH_OWNER);
54     NvFlushHierarchy(TPM_RH_ENDORSEMENT);

```

```
55
56 // Save hierarchy changes to NV
57 NvWriteReserved(NV_SP_SEED, &gp.SPSeed);
58 NvWriteReserved(NV_SH_PROOF, &gp.shProof);
59 NvWriteReserved(NV_EH_PROOF, &gp.ehProof);
60 NvWriteReserved(NV_OWNER_AUTH, &gp.ownerAuth);
61 NvWriteReserved(NV_ENDORSEMENT_AUTH, &gp.endorsementAuth);
62 NvWriteReserved(NV_LOCKOUT_AUTH, &gp.lockoutAuth);
63 NvWriteReserved(NV_OWNER_ALG, &gp.ownerAlg);
64 NvWriteReserved(NV_ENDORSEMENT_ALG, &gp.endorsementAlg);
65 NvWriteReserved(NV_OWNER_POLICY, &gp.ownerPolicy);
66 NvWriteReserved(NV_ENDORSEMENT_POLICY, &gp.endorsementPolicy);
67
68 // Initialize dictionary attack parameters
69 DAPreInstall_Init();
70
71 // Reset clock
72 go.clock = 0;
73 go.clockSafe = YES;
74 NvWriteReserved(NV_CLOCK, &go.clock);
75
76 // Reset counters
77 gp.resetCount = gr.restartCount = gr.clearCount = 0;
78 gp.auditCounter = 0;
79 NvWriteReserved(NV_RESET_COUNT, &gp.resetCount);
80 NvWriteReserved(NV_AUDIT_COUNTER, &gp.auditCounter);
81
82 // orderly state should be cleared because of the update to state clear data
83 g_clearOrderly = TRUE;
84
85 return TPM_RC_SUCCESS;
86 }
```

## 26.7 TPM2\_ClearControl

### 26.7.1 General Description

TPM2\_ClearControl() disables and enables the execution of TPM2\_Clear().

The TPM will SET the TPM's TPMA\_PERMANENT.*disableClear* attribute if *disable* is YES and will CLEAR the attribute if *disable* is NO. When the attribute is SET, TPM2\_Clear() may not be executed.

NOTE This is to simplify the logic of TPM2\_Clear(). TPM2\_ClearControl() can be called using *platformAuth* to CLEAR the *disableClear* attribute and then execute TPM2\_Clear().

*LockoutAuth* may be used to SET *disableClear* but not to CLEAR it.

*PlatformAuth* may be used to SET or CLEAR *disableClear*.

## 26.7.2 Command and Response

Table 159 — TPM2\_ClearControl Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ClearControl {NV}
TPMI_RH_CLEAR	@auth	TPM_RH_LOCKOUT or TPM_RH_PLATFORM+{PP} Auth Handle: 1 Auth Role: USER
TPMI_YES_NO	disable	YES if the <i>disableOwnerClear</i> flag is to be SET, NO if the flag is to be CLEAR.

Table 160 — TPM2\_ClearControl Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 26.7.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "ClearControl_fp.h"

```

Error Returns	Meaning
TPM_RC_AUTH_FAIL	authorization is not properly given

```

3  TPM_RC
4  TPM2_ClearControl(
5      ClearControl_In      *in          // IN: input parameter list
6  )
7  {
8      TPM_RC      result;
9
10     // The command needs NV update. Check if NV is available.
11     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
12     // this point
13     result = NvIsAvailable();
14     if(result != TPM_RC_SUCCESS) return result;
15
16     // Input Validation
17
18     // LockoutAuth may be used to set disableLockoutClear to TRUE but not to FALSE
19     if(in->auth == TPM_RH_LOCKOUT && in->disable == NO)
20         return TPM_RC_AUTH_FAIL;
21
22     // Internal Data Update
23
24     if(in->disable == YES)
25         gp.disableClear = TRUE;
26     else
27         gp.disableClear = FALSE;
28
29     // Record the change to NV
30     NvWriteReserved(NV_DISABLE_CLEAR, &gp.disableClear);
31
32     return TPM_RC_SUCCESS;
33 }

```

## 26.8 TPM2\_HierarchyChangeAuth

### 26.8.1 General Description

This command allows the authorization secret for a hierarchy or lockout to be changed using the current authorization value as the command authorization.

If *authHandle* is TPM\_RH\_PLATFORM, then *platformAuth* is changed. If *authHandle* is TPM\_RH\_OWNER, then *ownerAuth* is changed. If *authHandle* is TPM\_RH\_ENDORSEMENT, then *endorsementAuth* is changed. If *authHandle* is TPM\_RH\_LOCKOUT, then *lockoutAuth* is changed.

If *authHandle* is TPM\_RH\_PLATFORM, then Physical Presence may need to be asserted for this command to succeed (see 28.2, “TPM2\_PP\_Commands”).

The authorization value may be no larger than the digest produced by the hash algorithm used for context integrity.

EXAMPLE        If SHA384 is used in the computation of the integrity values for saved contexts, then the largest authorization value is 48 octets.



26.8.2 Command and Response

Table 161 — TPM2\_HierarchyChangeAuth Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_HierarchyChangeAuth {NV}
TPMI_RH_HIERARCHY_AUTH	@authHandle	TPM_RH_LOCKOUT, TPM_RH_ENDORSEMENT, TPM_RH_OWNER or TPM_RH_PLATFORM+{PP} Auth Index: 1 Auth Role: USER
TPM2B_AUTH	newAuth	new authorization secret

Table 162 — TPM2\_HierarchyChangeAuth Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 26.8.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "HierarchyChangeAuth_fp.h"
3  #include "Object_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_SIZE	<i>newAuth</i> size is greater than that of integrity hash digest

```

4  TPM_RC
5  TPM2_HierarchyChangeAuth(
6      HierarchyChangeAuth_In      *in          // IN: input parameter list
7  )
8  {
9      TPM_RC      result;
10
11     // The command needs NV update. Check if NV is available.
12     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
13     // this point
14     result = NvIsAvailable();
15     if(result != TPM_RC_SUCCESS) return result;
16
17     // Make sure the the auth value is a reasonable size (not larger than
18     // the size of the digest produced by the integrity hash. The integrity
19     // hash is assumed to produce the longest digest of any hash implemented
20     // on the TPM.
21     if( MemoryRemoveTrailingZeros(&in->newAuth)
22         > CryptGetHashDigestSize(CONTEXT_INTEGRITY_HASH_ALG))
23         return TPM_RC_SIZE + RC_HierarchyChangeAuth_newAuth;
24
25     // Set hierarchy authValue
26     switch(in->authHandle)
27     {
28     case TPM_RH_OWNER:
29         gp.ownerAuth = in->newAuth;
30         NvWriteReserved(NV_OWNER_AUTH, &gp.ownerAuth);
31         break;
32     case TPM_RH_ENDORSEMENT:
33         gp.endorsementAuth = in->newAuth;
34         NvWriteReserved(NV_ENDORSEMENT_AUTH, &gp.endorsementAuth);
35         break;
36     case TPM_RH_PLATFORM:
37         gc.platformAuth = in->newAuth;
38         // orderly state should be cleared
39         g_clearOrderly = TRUE;
40         break;
41     case TPM_RH_LOCKOUT:
42         gp.lockoutAuth = in->newAuth;
43         NvWriteReserved(NV_LOCKOUT_AUTH, &gp.lockoutAuth);
44         break;
45     default:
46         pAssert(FALSE);
47         break;
48     }
49
50     return TPM_RC_SUCCESS;
51 }

```

## 27 Dictionary Attack Functions

### 27.1 Introduction

A TPM is required to have support for logic that will help prevent a dictionary attack on an authorization value. The protection is provided by a counter that increments when a password authorization or an HMAC authorization fails. When the counter reaches a predefined value, the TPM will not accept, for some time interval, further requests that require authorization and the TPM is in Lockout mode. While the TPM is in Lockout mode, the TPM will return `TPM_RC_LOCKED` if the command requires use of an object's or Index's *authValue* unless the authorization applies to an entry in the Platform hierarchy.

**NOTE** Authorizations for objects and NV Index values in the Platform hierarchy are never locked out. However, a command that requires multiple authorizations will not be accepted when the TPM is in Lockout mode unless all of the authorizations reference objects and indexes in the Platform hierarchy.

If the TPM is continuously powered for the duration of *newRecoveryTime* and no authorization failures occur, the authorization failure counter will be decremented by one. This property is called "self-healing." Self-healing shall not cause the count of failed attempts to decrement below zero.

The count of failed attempts, the lockout interval, and self-healing interval are settable using `TPM2_DictionaryAttackParameters()`. The lockout parameters and the current value of the lockout counter can be read with `TPM2_GetCapability()`.

Dictionary attack protection does not apply to an entity associated with a permanent handle (handle type == `TPM_HT_PERMANENT`).

### 27.2 TPM2\_DictionaryAttackLockReset

#### 27.2.1 General Description

This command cancels the effect of a TPM lockout due to a number of successive authorization failures. If this command is properly authorized, the lockout counter is set to zero.

Only one authorization failure is allowed for this command during a *lockoutRecovery* interval (set using `TPM2_DictionaryAttackParameters()`).

## 27.2.2 Command and Response

Table 163 — TPM2\_DictionaryAttackLockReset Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_DictionaryAttackLockReset {NV}
TPMI_RH_LOCKOUT	@lockHandle	TPM_RH_LOCKOUT Auth Index: 1 Auth Role: USER

Table 164 — TPM2\_DictionaryAttackLockReset Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 27.2.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "DictionaryAttackLockReset_fp.h"
3  TPM_RC
4  TPM2_DictionaryAttackLockReset(
5      DictionaryAttackLockReset_In    *in           // IN: input parameter list
6  )
7  {
8      TPM_RC          result;
9
10     // Input parameter is not reference in command action
11     in = NULL;
12
13     // The command needs NV update. Check if NV is available.
14     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
15     // this point
16     result = NvIsAvailable();
17     if(result != TPM_RC_SUCCESS) return result;
18
19     // Internal Data Update
20
21     // Set failed tries to 0
22     gp.failedTries = 0;
23
24     // Record the changes to NV
25     NvWriteReserved(NV_FAILED_TRIES, &gp.failedTries);
26
27     return TPM_RC_SUCCESS;
28 }
```

## 27.3 TPM2\_DictionaryAttackParameters

### 27.3.1 General Description

This command changes the lockout parameters.

The command requires *lockoutAuth*.

The timeout parameters (*newRecoveryTime* and *lockoutRecovery*) indicate values that are measured with respect to the *Time* and not *Clock*.

NOTE            Use of *Time* means that the TPM shall be continuously powered for the duration of a timeout.

If *newRecoveryTime* is zero, then DA protection is disabled. Authorizations are checked but authorization failures will not cause the TPM to enter lockout.

If *newMaxTries* is zero, the TPM will be in lockout and use of DA protected entities will be disabled.

If *lockoutRecovery* is zero, then the recovery interval is a boot cycle (*\_TPM\_Init* followed by *Startup(CLEAR)*).

This command will set the authorization failure count (*failedTries*) to zero.

Only one authorization failure is allowed for this command during a *lockoutRecovery* interval.

27.3.2 Command and Response

Table 165 — TPM2\_DictionaryAttackParameters Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_DictionaryAttackParameters {NV}
TPMI_RH_LOCKOUT	@lockHandle	TPM_RH_LOCKOUT Auth Index: 1 Auth Role: USER
UINT32	newMaxTries	count of authorization failures before the lockout is imposed
UINT32	newRecoveryTime	time in seconds before the authorization failure count is automatically decremented A value of zero indicates that DA protection is disabled.
UINT32	lockoutRecovery	time in seconds after a <i>lockoutAuth</i> failure before use of <i>lockoutAuth</i> is allowed A value of zero indicates that a reboot is required.

Table 166 — TPM2\_DictionaryAttackParameters Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 27.3.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "DictionaryAttackParameters_fp.h"
3  TPM_RC
4  TPM2_DictionaryAttackParameters(
5      DictionaryAttackParameters_In  *in          // IN: input parameter list
6  )
7  {
8      TPM_RC      result;
9
10     // The command needs NV update. Check if NV is available.
11     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
12     // this point
13     result = NvIsAvailable();
14     if(result != TPM_RC_SUCCESS) return result;
15
16     // Internal Data Update
17
18     // Set dictionary attack parameters
19     gp.maxTries = in->newMaxTries;
20     gp.recoveryTime = in->newRecoveryTime;
21     gp.lockoutRecovery = in->lockoutRecovery;
22
23     // Set failed tries to 0
24     gp.failedTries = 0;
25
26     // Record the changes to NV
27     NvWriteReserved(NV_FAILED_TRIES, &gp.failedTries);
28     NvWriteReserved(NV_MAX_TRIES, &gp.maxTries);
29     NvWriteReserved(NV_RECOVERY_TIME, &gp.recoveryTime);
30     NvWriteReserved(NV_LOCKOUT_RECOVERY, &gp.lockoutRecovery);
31
32     return TPM_RC_SUCCESS;
33 }
```



## 28 Miscellaneous Management Functions

### 28.1 Introduction

This clause contains commands that do not logically group with any other commands.

### 28.2 TPM2\_PP\_Commands

#### 28.2.1 General Description

This command is used to determine which commands require assertion of Physical Presence (PP) in addition to *platformAuth/platformPolicy*.

This command requires that *auth* is TPM\_RH\_PLATFORM and that Physical Presence be asserted.

After this command executes successfully, the commands listed in *setList* will be added to the list of commands that require that Physical Presence be asserted when the handle associated with the authorization is TPM\_RH\_PLATFORM. The commands in *clearList* will no longer require assertion of Physical Presence in order to authorize a command.

If a command is not in either list, its state is not changed. If a command is in both lists, then it will no longer require Physical Presence (for example, *setList* is processed first).

Only commands with handle types of TPML\_RH\_PLATFORM, TPML\_RH\_PROVISION, TPML\_RH\_CLEAR, or TPML\_RH\_HIERARCHY can be gated with Physical Presence. If any other command is in either list, it is discarded.

When a command requires that Physical Presence be provided, then Physical Presence shall be asserted for either an HMAC or a Policy authorization.

NOTE Physical Presence may be made a requirement of any policy.

TPM2\_PP\_Commands() always requires assertion of Physical Presence.

## 28.2.2 Command and Response

Table 167 — TPM2\_PP\_Commands Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PP_Commands {NV}
TPMI_RH_PLATFORM	@auth	TPM_RH_PLATFORM+PP Auth Index: 1 Auth Role: USER + Physical Presence
TPML_CC	setList	list of commands to be added to those that will require that Physical Presence be asserted
TPML_CC	clearList	list of commands that will no longer require that Physical Presence be asserted

Table 168 — TPM2\_PP\_Commands Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 28.2.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "PP_Commands_fp.h"
3  TPM_RC
4  TPM2_PP_Commands(
5      PP_Commands_In  *in           // IN: input parameter list
6  )
7  {
8      UINT32          i;
9
10     TPM_RC          result;
11
12     // The command needs NV update. Check if NV is available.
13     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
14     // this point
15     result = NvIsAvailable();
16     if(result != TPM_RC_SUCCESS) return result;
17
18     // Internal Data Update
19
20     // Process set list
21     for(i = 0; i < in->setList.count; i++)
22         // If command is implemented, set it as PP required. If the input
23         // command is not a PP command, it will be ignored at
24         // PhysicalPresenceCommandSet().
25         if(CommandIsImplemented(in->setList.commandCodes[i]))
26             PhysicalPresenceCommandSet(in->setList.commandCodes[i]);
27
28     // Process clear list
29     for(i = 0; i < in->clearList.count; i++)
30         // If command is implemented, clear it as PP required. If the input
31         // command is not a PP command, it will be ignored at
32         // PhysicalPresenceCommandClear(). If the input command is
33         // TPM2_PP_Commands, it will be ignored as well
34         if(CommandIsImplemented(in->clearList.commandCodes[i]))
35             PhysicalPresenceCommandClear(in->clearList.commandCodes[i]);
36
37     // Save the change of PP list
38     NvWriteReserved(NV_PP_LIST, &gp.ppList);
39
40     return TPM_RC_SUCCESS;
41 }

```

## 28.3 TPM2\_SetAlgorithmSet

### 28.3.1 General Description

This command allows the platform to change the set of algorithms that are used by the TPM. The *algorithmSet* setting is a vendor-dependent value.

If the changing of the algorithm set results in a change of the algorithms of PCR banks, then the TPM will need to be reset (`_TPM_Init` and `TPM2_Startup(TPM_SU_CLEAR)`) before the new PCR settings take effect. After this command executes successfully, if *startupType* in the next `TPM2_Startup()` is not `TPM_SU_CLEAR`, the TPM shall return `TPM_RC_VALUE` and enter Failure mode.

This command does not change the algorithms available to the platform.

NOTE            The reference implementation does not have support for this command. In particular, it does not support use of this command to selectively disable algorithms. Proper support would require modification of the unmarshaling code so that each time an algorithm is unmarshaled, it would be verified as being enabled.

## 28.3.2 Command and Response

Table 169 — TPM2\_SetAlgorithmSet Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_SetAlgorithmSet {NV}
TPMI_RH_PLATFORM	@authHandle	TPM_RH_PLATFORM Auth Index: 1 Auth Role: USER
UINT32	algorithmSet	a TPM vendor-dependent value indicating the algorithm set selection

Table 170 — TPM2\_SetAlgorithmSet Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 28.3.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "SetAlgorithmSet_fp.h"
3  TPM_RC
4  TPM2_SetAlgorithmSet(
5      SetAlgorithmSet_In    *in           // IN: input parameter list
6  )
7  {
8      TPM_RC    result;
9
10     // The command needs NV update. Check if NV is available.
11     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
12     // this point
13     result = NvIsAvailable();
14     if(result != TPM_RC_SUCCESS) return result;
15
16     // Internal Data Update
17     gp.algorithmSet = in->algorithmSet;
18
19     // Write the algorithm set changes to NV
20     NvWriteReserved(NV_ALGORITHM_SET, &gp.algorithmSet);
21
22     return TPM_RC_SUCCESS;
23 }
```

## 29 Field Upgrade

### 29.1 Introduction

This clause contains the commands for managing field upgrade of the firmware in the TPM. The field upgrade scheme may be used for replacement or augmentation of the firmware installed in the TPM.

EXAMPLE 1 If an algorithm is found to be flawed, a patch of that algorithm might be installed using the firmware upgrade process. The patch might be a replacement of a portion of the code or a complete replacement of the firmware.

EXAMPLE 2 If an additional set of ECC parameters is needed, the firmware process may be used to add the parameters to the TPM data set.

The field upgrade process uses two commands (TPM2\_FieldUpgradeStart() and TPM2\_FieldUpgradeData()). TPM2\_FieldUpgradeStart() validates that a signature on the provided digest is from the TPM manufacturer and that proper authorization is provided using *platformPolicy*.

NOTE 1 The *platformPolicy* for field upgraded is defined by the PM and may include requirements that the upgrade be signed by the PM or the TPM owner and include any other constraints that are desired by the PM.

If the proper authorization is given, the TPM will retain the signed digest and enter the Field Upgrade mode (FUM). While in FUM, the TPM will accept TPM2\_FieldUpgradeData() commands. It may accept other commands if it is able to complete them using the previously installed firmware. Otherwise, it will return TPM\_RC\_UPGRADE.

Each block of the field upgrade shall contain the digest of the next block of the field upgrade data. That digest shall be included in the digest of the previous block. The digest of the first block is signed by the TPM manufacturer. That signature and first block digest are the parameters for TPM2\_FieldUpgradeStart(). The digest is saved in the TPM as the required digest for the next field upgrade data block and as the identifier of the field upgrade sequence.

For each field upgrade data block that is sent to the TPM by TPM2\_FieldUpgradeData(), the TPM shall validate that the digest matches the required digest and if not, shall return TPM\_RC\_VALUE. The TPM shall extract the digest of the next expected block and return that value to the caller, along with the digest of the first data block of the update sequence.

The system may attempt to abandon the firmware upgrade by using a zero-length buffer in TPM2\_FieldUpgradeData(). If the TPM is able to resume operation using the firmware present when the upgrade started, then the TPM will indicate that it has abandon the update by setting the digest of the next block to the Empty Buffer. If the TPM cannot abandon the update, it will return the expected next digest.

The system may also attempt to abandon the update because of a power interruption. If the TPM is able to resume normal operations, then it will respond normally to TPM2\_Startup(). If the TPM is not able to resume normal operations, then it will respond to any command but TPM2\_FieldUpgradeData() with TPM\_RC\_FIELDUPGRADE.

After a \_TPM\_Init, system software may not be able to resume the field upgrade that was in process when the power interruption occurred. In such case, the TPM firmware may be reset to one of two other values:

- the original firmware that was installed at the factory (“initial firmware”); or
- the firmware that was in the TPM when the field upgrade process started (“previous firmware”).

The TPM retains the digest of the first block for these firmware images and checks to see if the first block after \_TPM\_Init matches either of those digests. If so, the firmware update process restarts and the original firmware may be loaded.

NOTE 2           The TPM is required to accept the previous firmware as either a vendor-provided update or as recovered from the TPM using TPM2\_FirmwareRead().

When the last block of the firmware upgrade is loaded into the TPM (indicated to the TPM by data in the data block in a TPM vendor-specific manner), the TPM will complete the upgrade process. If the TPM is able to resume normal operations without a reboot, it will set the hash algorithm of the next block to TPM\_ALG\_NULL and return TPM\_RC\_SUCCESS. If a reboot is required, the TPM shall return TPM\_RC\_REBOOT in response to the last TPM2\_FieldUpgradeData() and all subsequent TPM commands until a \_TPM\_Init is received.

NOTE 3           Because no additional data is allowed when the response code is not TPM\_RC\_SUCCESS, the TPM returns TPM\_RC\_SUCCESS for all calls to TPM2\_FieldUpgradeData() except the last. In this manner, the TPM is able to indicate the digest of the next block. If a \_TPM\_Init occurs while the TPM is in FUM, the next block may be the digest for the first block of the original firmware. If it is not, then the TPM will not accept the original firmware until the next \_TPM\_Init when the TPM is in FUM.

During the field upgrade process, the TPM shall preserve:

- Primary Seeds;
- Hierarchy *authValue*, *authPolicy*, and *proof* values;
- Lockout *authValue* and authorization failure count values;
- PCR *authValue* and *authPolicy* values;
- NV Index allocations and contents;
- Persistent object allocations and contents; and
- Clock.



## 29.2 TPM2\_FieldUpgradeStart

### 29.2.1 General Description

This command uses *platformPolicy* and a TPM Vendor Authorization Key to authorize a Field Upgrade Manifest.

If the signature checks succeed, the authorization is valid and the TPM will accept TPM2\_FieldUpgradeData().

This signature is checked against the loaded key referenced by *keyHandle*. This key will have a Name that is the same as a value that is part of the TPM firmware data. If the signature is not valid, the TPM shall return TPM\_RC\_SIGNATURE.

NOTE            A loaded key is used rather than a hard-coded key to reduce the amount of memory needed for this key data in case more than one vendor key is needed.

## 29.2.2 Command and Response

Table 171 — TPM2\_FieldUpgradeStart Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_FieldUpgradeStart
TPMI_RH_PLATFORM	@authorization	TPM_RH_PLATFORM+{PP} Auth Index:1 Auth Role: ADMIN
TPMI_DH_OBJECT	keyHandle	handle of a public area that contains the TPM Vendor Authorization Key that will be used to validate <i>manifestSignature</i> Auth Index: None
TPM2B_DIGEST	fuDigest	digest of the first block in the field upgrade sequence
TPMT_SIGNATURE	manifestSignature	signature over <i>fuDigest</i> using the key associated with <i>keyHandle</i> (not optional)

Table 172 — TPM2\_FieldUpgradeStart Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

### 29.2.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "FieldUpgradeStart_fp.h"
3  #if CC_FieldUpgradeStart == YES
4  TPM_RC
5  TPM2_FieldUpgradeStart(
6      FieldUpgradeStart_In      *in          // IN: input parameter list
7  )
8  {
9      // Not implemented
10     UNUSED_PARAMETER(in);
11     return TPM_RC_SUCCESS;
12 }
13 #endif
```

### 29.3 TPM2\_FieldUpgradeData

#### 29.3.1 General Description

This command will take the actual field upgrade image to be installed on the TPM. The exact format of *fuData* is vendor-specific. This command is only possible following a successful TPM2\_FieldUpgradeStart(). If the TPM has not received a properly authorized TPM2\_FieldUpgradeStart(), then the TPM shall return TPM\_RC\_FIELDUPGRADE.

The TPM will validate that the digest of *fuData* matches an expected value. If so, the TPM may buffer or immediately apply the update. If the digest of *fuData* does not match an expected value, the TPM shall return TPM\_RC\_VALUE.

## 29.3.2 Command and Response

Table 173 — TPM2\_FieldUpgradeData Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_FieldUpgradeData {NV}
TPM2B_MAX_BUFFER	fuData	field upgrade image data

Table 174 — TPM2\_FieldUpgradeData Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPMT_HA+	nextDigest	tagged digest of the next block TPM_ALG_NULL if field update is complete
TPMT_HA	firstDigest	tagged digest of the first block of the sequence

### 29.3.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "FieldUpgradeData_fp.h"
3  #if CC_FieldUpgradeData == YES
4  TPM_RC
5  TPM2_FieldUpgradeData(
6      FieldUpgradeData_In      *in,          // IN: input parameter list
7      FieldUpgradeData_Out     *out         // OUT: output parameter list
8  )
9  {
10     // Not implemented
11     UNUSED_PARAMETER(in);
12     UNUSED_PARAMETER(out);
13     return TPM_RC_SUCCESS;
14 }
15 #endif
```

## 29.4 TPM2\_FirmwareRead

### 29.4.1 General Description

This command is used to read a copy of the current firmware installed in the TPM.

The presumption is that the data will be returned in reverse order so that the last block in the sequence would be the first block given to the TPM in case of a failure recovery. If the TPM2\_FirmwareRead sequence completes successfully, then the data provided from the TPM will be sufficient to allow the TPM to recover from an abandoned upgrade of this firmware.

To start the sequence of retrieving the data, the caller sets *sequenceNumber* to zero. When the TPM has returned all the firmware data, the TPM will return the Empty Buffer as *fuData*.

The contents of *fuData* are opaque to the caller.

NOTE 1            The caller should retain the ordering of the update blocks so that the blocks sent to the TPM have the same size and inverse order as the blocks returned by a sequence of calls to this command.

NOTE 2            Support for this command is optional even if the TPM implements TPM2\_FieldUpgradeStart() and TPM2\_FieldUpgradeData().

## 29.4.2 Command and Response

Table 175 — TPM2\_FirmwareRead Command

Type	Name	Description
TPML_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_FirmwareRead
UINT32	sequenceNumber	the number of previous calls to this command in this sequence set to 0 on the first call

Table 176 — TPM2\_FirmwareRead Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_MAX_BUFFER	fuData	field upgrade image data



### 29.4.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "FirmwareRead_fp.h"
3  TPM_RC
4  TPM2_FirmwareRead(
5      FirmwareRead_In      *in,          // IN: input parameter list
6      FirmwareRead_Out     *out         // OUT: output parameter list
7  )
8  {
9      // Not implemented
10     UNUSED_PARAMETER(in);
11     UNUSED_PARAMETER(out);
12     return TPM_RC_SUCCESS;
13 }
```

## 30 Context Management

### 30.1 Introduction

Three of the commands in this clause (TPM2\_ContextSave(), TPM2\_ContextLoad(), and TPM2\_FlushContext()) implement the resource management described in the "Context Management" clause in Part 1.

The fourth command in this clause (TPM2\_EvictControl()) is used to control the persistence of a loadable objects in TPM memory. Background for this command may be found in the "Owner and Platform Evict Objects" clause in Part 1.

### 30.2 TPM2\_ContextSave

#### 30.2.1 General Description

This command saves a session context, object context, or sequence object context outside the TPM.

No authorization sessions of any type are allowed with this command and tag is required to be TPM\_ST\_NO\_SESSIONS.

**NOTE** This preclusion avoids complex issues of dealing with the same session in *handle* and in the session area. While it might be possible to provide specificity, it would add unnecessary complexity to the TPM and, because this capability would provide no application benefit, use of authorization sessions for audit or encryption is prohibited.

The TPM shall encrypt and integrity protect the context as described in the "Context Protection" clause in Part 1.

See the "Context Data" clause in Part 2 for a description of the *context* structure in the response.

## 30.2.2 Command and Response

Table 177 — TPM2\_ContextSave Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ContextSave
TPMI_DH_CONTEXT	saveHandle	handle of the resource to save Auth Index: None

Table 178 — TPM2\_ContextSave Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPMS_CONTEXT	context	

## 30.2.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "ContextSave_fp.h"
3  #include "Context_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_CONTEXT_GAP	a <i>contextID</i> could not be assigned for a session context save
TPM_RC_TOO_MANY_CONTEXTS	no more contexts can be saved as the counter has maxed out

```

4  TPM_RC
5  TPM2_ContextSave(
6      ContextSave_In    *in,        // IN: input parameter list
7      ContextSave_Out  *out        // OUT: output parameter list
8  )
9  {
10     TPM_RC              result;
11     UINT16              fingerprintSize; // The size of fingerprint in context
12     // blob.
13     UINT64              contextID = 0; // session context ID
14     TPM2B_SYM_KEY       symKey;
15     TPM2B_IV            iv;
16
17     TPM2B_DIGEST        integrity;
18     UINT16              integritySize;
19     BYTE                *buffer;
20
21     // This command may cause the orderlyState to be cleared due to
22     // the update of state reset data. If this is the case, check if NV is
23     // available first
24     if(gp.orderlyState != SHUTDOWN_NONE)
25     {
26         // The command needs NV update. Check if NV is available.
27         // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
28         // this point
29         result = NvIsAvailable();
30         if(result != TPM_RC_SUCCESS) return result;
31     }
32
33     // Internal Data Update
34
35     // Initialize output handle. At the end of command action, the output
36     // handle of an object will be replaced, while the output handle
37     // for a session will be the same as input
38     out->context.savedHandle = in->saveHandle;
39
40     // Get the size of fingerprint in context blob. The sequence value in
41     // TPMS_CONTEXT structure is used as the fingerprint
42     fingerprintSize = sizeof(out->context.sequence);
43
44     // Compute the integrity size at the beginning of context blob
45     integritySize = sizeof(integrity.t.size)
46                   + CryptGetHashDigestSize(CONTEXT_INTEGRITY_HASH_ALG);
47
48
49     // Perform object or session specific context save
50     switch(HandleGetType(in->saveHandle))
51     {
52     case TPM_HT_TRANSIENT:
53     {

```

```

54     OBJECT          *object = ObjectGet(in->saveHandle);
55
56     // Set size of the context data. The contents of context blob is vendor
57     // defined. In this implementation, the size is size of integrity
58     // plus fingerprint plus the whole internal OBJECT structure
59     out->context.contextBlob.t.size = integritySize +
60         fingerprintSize + sizeof(*object);
61
62     // Copy the whole internal OBJECT structure to context blob, leave
63     // the size for fingerprint
64     MemoryCopy(out->context.contextBlob.t.buffer
65         + integritySize + fingerprintSize,
66         object, sizeof(*object));
67
68     // Increment object context ID
69     gr.objectContextID++;
70     // If object context ID overflows, TPM should be put in failure mode
71     if(gr.objectContextID == 0)
72         FAIL(FATAL_ERROR_INTERNAL);
73
74     // Fill in other return values for an object.
75     out->context.sequence = gr.objectContextID;
76     // For regular object, savedHandle is 0x80000000. For sequence object,
77     // savedHandle is 0x80000001. For object with stClear, savedHandle
78     // is 0x80000002
79     if(ObjectIsSequence(object))
80     {
81         out->context.savedHandle = 0x80000001;
82     }
83     else if(object->attributes.stClear == SET)
84     {
85         out->context.savedHandle = 0x80000002;
86     }
87     else
88     {
89         out->context.savedHandle = 0x80000000;
90     }
91
92     // Get object hierarchy
93     out->context.hierarchy = ObjectDataGetHierarchy(object);
94
95     break;
96 }
97 case TPM_HT_HMAC_SESSION:
98 case TPM_HT_POLICY_SESSION:
99 {
100     SESSION          *session = SessionGet(in->saveHandle);
101
102     // Set size of the context data. The contents of context blob is vendor
103     // defined. In this implementation, the size of context blob is the
104     // size of a internal session structure plus the size of
105     // fingerprint plus the size of integrity
106     out->context.contextBlob.t.size = integritySize +
107         fingerprintSize + sizeof(*session);
108
109     // Copy the whole internal SESSION structure to context blob.
110     // Save space for fingerprint at the beginning of the buffer
111     // This is done before anything else so that the actual context
112     // can be reclaimed after this call
113     MemoryCopy(out->context.contextBlob.t.buffer
114         + integritySize + fingerprintSize,
115         session, sizeof(*session));
116
117     // Fill in the other return parameters for a session

```

```
118     // Get a context ID and set the session tracking values appropriately
119     // TPM_RC_CONTEXT_GAP is a possible error.
120     // SessionContextSave() will flush the in-memory context
121     // so no additional errors may occur after this call.
122     result = SessionContextSave(out->context.savedHandle, &contextID);
123     if(result != TPM_RC_SUCCESS) return result;
124
125     // sequence number is the current session contextID
126     out->context.sequence = contextID;
127
128     // use TPM_RH_NULL as hierarchy for session context
129     out->context.hierarchy = TPM_RH_NULL;
130
131     break;
132 }
133 default:
134     // SaveContext may only take an object handle or a session handle.
135     // All the other handle type should be filtered out at unmarshal
136     pAssert(FALSE);
137     break;
138 }
139
140 // Save fingerprint at the beginning of encrypted area of context blob.
141 // Reserve the integrity space
142 MemoryCopy(out->context.contextBlob.t.buffer + integritySize,
143           &out->context.sequence, sizeof(out->context.sequence));
144
145 // Compute context encryption key
146 ComputeContextProtectionKey(&out->context, &symKey, &iv);
147
148 // Encrypt context blob
149 CryptSymmetricEncrypt(out->context.contextBlob.t.buffer + integritySize,
150                      CONTEXT_ENCRYPT_ALG, CONTEXT_ENCRYPT_KEY_BITS,
151                      TPM_ALG_CFB, symKey.t.buffer, &iv,
152                      out->context.contextBlob.t.size - integritySize,
153                      out->context.contextBlob.t.buffer + integritySize);
154
155 // Compute integrity hash for the object
156 // In this implementation, the same routine is used for both sessions
157 // and objects.
158 ComputeContextIntegrity(&out->context, &integrity);
159
160 // add integrity at the beginning of context blob
161 buffer = out->context.contextBlob.t.buffer;
162 TPM2B_DIGEST_Marshal(&integrity, &buffer, NULL);
163
164 // orderly state should be cleared because of the update of state reset and
165 // state clear data
166 g_clearOrderly = TRUE;
167
168 return TPM_RC_SUCCESS;
169 }
```

### 30.3 TPM2\_ContextLoad

#### 30.3.1 General Description

This command is used to reload a context that has been saved by TPM2\_ContextSave().

No authorization sessions of any type are allowed with this command and tag is required to be TPM\_ST\_NO\_SESSIONS (see note in 30.2.1).

The TPM will return TPM\_RC\_HIERARCHY if the context is associated with a hierarchy that is disabled.

NOTE                      Contexts for authorization sessions and for sequence objects belong to the NULL hierarchy which is never disabled.

See the "Context Data" clause in Part 2 for a description of the values in the *context* parameter.

If the integrity HMAC of the saved context is not valid, the TPM shall return TPM\_RC\_INTEGRITY.

The TPM shall perform a check on the decrypted context as described in the "Context Confidentiality Protections" clause of Part 1 and enter failure mode if the check fails.

## 30.3.2 Command and Response

Table 179 — TPM2\_ContextLoad Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ContextLoad
TPMS_CONTEXT	context	the context blob

Table 180 — TPM2\_ContextLoad Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPMI_DH_CONTEXT	loadedHandle	the handle assigned to the resource after it has been successfully loaded



## 30.3.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "ContextLoad_fp.h"
3  #include "Context_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_CONTEXT_GAP	there is only one available slot and this is not the oldest saved session context
TPM_RC_HANDLE	'context.savedHandle' does not reference a saved session
TPM_RC_HIERARCHY	'context.hierarchy' is disabled
TPM_RC_INTEGRITY	context integrity check fail
TPM_RC_OBJECT_MEMORY	no free slot for an object
TPM_RC_SESSION_MEMORY	no free session slots
TPM_RC_SIZE	incorrect context blob size

```

4  TPM_RC
5  TPM2_ContextLoad(
6      ContextLoad_In    *in,          // IN: input parameter list
7      ContextLoad_Out  *out          // OUT: output parameter list
8  )
9  {
10     // Local Variables
11     TPM_RC      result = TPM_RC_SUCCESS;
12
13     TPM2B_DIGEST    ingerityToCompare;
14     TPM2B_DIGEST    integrity;
15     UINT16          integritySize;
16     UINT64          fingerprint;
17     BYTE            *buffer;
18     INT32           size;
19
20     TPM_HT          handleType;
21     TPM2B_SYM_KEY   symKey;
22     TPM2B_IV        iv;
23
24     // Input Validation
25
26     // Check context blob size
27     handleType = HandleGetType(in->context.savedHandle);
28
29     // Check integrity
30     // In this implementation, the same routine is used for both sessions
31     // and objects.
32     integritySize = sizeof(integrity.t.size)
33                   + CryptGetHashDigestSize(CONTEXT_INTEGRITY_HASH_ALG);
34
35     // Get integrity from context blob
36     buffer = in->context.contextBlob.t.buffer;
37     size = (INT32) in->context.contextBlob.t.size;
38     result = TPM2B_DIGEST_Unmarshal(&integrity, &buffer, &size);
39     if(result != TPM_RC_SUCCESS)
40         return result;
41
42     // Compute context integrity
43     ComputeContextIntegrity(&in->context, &ingerityToCompare);

```

```

44
45 // Compare integrity
46 if(!Memory2BEqual(&integrity.b, &ingerityToCompare.b))
47     return TPM_RC_INTEGRITY + RC_ContextLoad_context;
48
49 // Compute context encryption key
50 ComputeContextProtectionKey(&in->context, &symKey, &iv);
51
52 // Decrypt context data in place
53 CryptSymmetricDecrypt(in->context.contextBlob.t.buffer + integritySize,
54     CONTEXT_ENCRYPT_ALG, CONTEXT_ENCRYPT_KEY_BITS,
55     TPM_ALG_CFB, symKey.t.buffer, &iv,
56     in->context.contextBlob.t.size - integritySize,
57     in->context.contextBlob.t.buffer + integritySize);
58
59 // Read the fingerprint value, skip the leading integrity size
60 MemoryCopy(&fingerprint, in->context.contextBlob.t.buffer + integritySize,
61     sizeof(fingerprint));
62 // Check fingerprint. If the check fails, TPM should be put to failure mode
63 if(fingerprint != in->context.sequence)
64     FAIL(FATAL_ERROR_INTERNAL);
65
66 // Perform object or session specific input check
67 switch(handleType)
68 {
69 case TPM_HT_TRANSIENT:
70 {
71     OBJECT    object;
72
73     // Discard any changes to the handle that the TRM might have made
74     in->context.savedHandle = TRANSIENT_FIRST;
75     // Get a copy of the object data in input context blob, skip the
76     // integrity and fingerprint area
77     MemoryCopy(&object, in->context.contextBlob.t.buffer +
78         integritySize + sizeof(fingerprint),
79         sizeof(object));
80
81     // If hierarchy is disabled, no object context can be loaded in this
82     // hierarchy
83     if(!HierarchyIsEnabled(in->context.hierarchy))
84         return TPM_RC_HIERARCHY + RC_ContextLoad_context;
85
86     // Restore object. A TPM_RC_OBJECT_MEMORY error may be returned at
87     // this point
88     result = ObjectContextLoad(&object, &out->loadedHandle);
89     if(result != TPM_RC_SUCCESS) return result;
90
91     break;
92 }
93 case TPM_HT_POLICY_SESSION:
94 case TPM_HT_HMAC_SESSION:
95 {
96
97     SESSION    session;
98
99     // This command may cause the orderlyState to be cleared due to
100    // the update of state reset data. If this is the case, check if NV is
101    // available first
102    if(gp.orderlyState != SHUTDOWN_NONE)
103    {
104        // The command needs NV update. Check if NV is available.
105        // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned
106        // at this point
107        result = NvIsAvailable();

```

```
108         if(result != TPM_RC_SUCCESS) return result;
109     }
110
111     // Check if input handle points to a valid saved session
112     if(!SessionIsSaved(in->context.savedHandle))
113         return TPM_RC_HANDLE + RC_ContextLoad_context;
114
115     // Retrieve session data from input context blob, skip the
116     // integrity and fingerprint area
117     MemoryCopy(&session, in->context.contextBlob.t.buffer +
118         integritySize + sizeof(fingerprint),
119         sizeof(session));
120
121     // Restore session. A TPM_RC_SESSION_MEMORY, TPM_RC_CONTEXT_GAP error
122     // may be returned at this point
123     result = SessionContextLoad(&session, &in->context.savedHandle);
124     if(result != TPM_RC_SUCCESS) return result;
125
126     out->loadedHandle = in->context.savedHandle;
127
128     // orderly state should be cleared because of the update of state
129     // reset and state clear data
130     g_clearOrderly = TRUE;
131
132     break;
133 }
134 default:
135     // Context blob may only have an object handle or a session handle.
136     // All the other handle type should be filtered out at unmarshal
137     pAssert(FALSE);
138     break;
139 }
140
141 return TPM_RC_SUCCESS;
142 }
```

## 30.4 TPM2\_FlushContext

### 30.4.1 General Description

This command causes all context associated with a loaded object or session to be removed from TPM memory.

This command may not be used to remove a persistent object from the TPM.

A session does not have to be loaded in TPM memory to have its context flushed. The saved session context associated with the indicated handle is invalidated.

No sessions of any type are allowed with this command and tag is required to be TPM\_ST\_NO\_SESSIONS (see note in 30.2.1).

If the handle is for a transient object and the handle is not associated with a loaded object, then the TPM shall return TPM\_RC\_HANDLE.

If the handle is for an authorization session and the handle does not reference a loaded or active session, then the TPM shall return TPM\_RC\_HANDLE.

## 30.4.2 Command and Response

Table 181 — TPM2\_FlushContext Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_FlushContext
TPMI_DH_CONTEXT	flushHandle	the handle of the item to flush NOTE This is a use of a handle as a parameter.

Table 182 — TPM2\_FlushContext Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 30.4.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "FlushContext_fp.h"

```

Error Returns	Meaning
TPM_RC_HANDLE	<i>flushHandle</i> does not reference a loaded object or session

```

3  TPM_RC
4  TPM2_FlushContext(
5      FlushContext_In      *in          // IN: input parameter list
6  )
7  {
8  // Internal Data Update
9
10     // Call object or session specific routine to flush
11     switch(HandleGetType(in->flushHandle))
12     {
13     case TPM_HT_TRANSIENT:
14         if(!ObjectIsPresent(in->flushHandle))
15             return TPM_RC_HANDLE;
16         // Flush object
17         ObjectFlush(in->flushHandle);
18         break;
19     case TPM_HT_HMAC_SESSION:
20     case TPM_HT_POLICY_SESSION:
21         if( !SessionIsLoaded(in->flushHandle)
22             && !SessionIsSaved(in->flushHandle)
23             )
24             return TPM_RC_HANDLE;
25
26         // If the session to be flushed is the exclusive audit session, then
27         // indicate that there is no exclusive audit session any longer.
28         if(in->flushHandle == g_exclusiveAuditSession)
29             g_exclusiveAuditSession = TPM_RH_UNASSIGNED;
30
31         // Flush session
32         SessionFlush(in->flushHandle);
33         break;
34     default:
35         // This command only take object or session handle.  Other handles
36         // should be filtered out at handle unmarshal
37         pAssert(FALSE);
38         break;
39     }
40
41     return TPM_RC_SUCCESS;
42 }

```

## 30.5 TPM2\_EvictControl

### 30.5.1 General Description

This command allows a transient object to be made persistent or a persistent object to be evicted.

NOTE 1 A transient object is one that may be removed from TPM memory using either TPM2\_FlushContext or TPM2\_Startup(). A persistent object is not removed from TPM memory by TPM2\_FlushContext() or TPM2\_Startup().

If *objectHandle* is a transient object, then the call is to make the object persistent and assign *persistentHandle* to the persistent version of the object. If *objectHandle* is a persistent object, then the call is to evict the persistent object.

Before execution of TPM2\_EvictControl code below, the TPM verifies that *objectHandle* references an object that is resident on the TPM and that *persistentHandle* is a valid handle for a persistent object.

NOTE 2 This requirement simplifies the unmarshaling code so that it only need check that *persistentHandle* is always a persistent object.

If *objectHandle* references a transient object:

- a) The TPM shall return TPM\_RC\_ATTRIBUTES if
  - 1) it is in the hierarchy of TPM\_RH\_NULL,
  - 2) only the public portion of the object is loaded, or
  - 3) the *stClear* is SET in the object or in an ancestor key.
- b) The TPM shall return TPM\_RC\_HIERARCHY if the object is not in the proper hierarchy as determined by *auth*.
  - 1) If *auth* is TPM\_RH\_PLATFORM, the proper hierarchy is the Platform hierarchy.
  - 2) If *auth* is TPM\_RH\_OWNER, the proper hierarchy is either the Storage or the Endorsement hierarchy.
- c) The TPM shall return TPM\_RC\_RANGE if *persistentHandle* is not in the proper range as determined by *auth*.
  - 1) If *auth* is TPM\_RH\_OWNER, then *persistentHandle* shall be in the inclusive range of 81 00 00 00<sub>16</sub> to 81 7F FF FF<sub>16</sub>.
  - 2) If *auth* is TPM\_RH\_PLATFORM, then *persistentHandle* shall be in the inclusive range of 81 80 00 00<sub>16</sub> to 81 FF FF FF<sub>16</sub>.
- d) The TPM shall return TPM\_RC\_NV\_DEFINED if a persistent object exists with the same handle as *persistentHandle*.
- e) The TPM shall return TPM\_RC\_NV\_SPACE if insufficient space is available to make the object persistent.
- f) The TPM shall return TPM\_RC\_NV\_SPACE if execution of this command will prevent the TPM from being able to hold two transient objects of any kind.

NOTE 3 This requirement anticipates that a TPM may be implemented such that all TPM memory is non-volatile and not subject to endurance issues. In such case, there is no movement of an object between memory of different types and it is necessary that the TPM ensure that it is always possible for the management software to move objects to/from TPM memory in order to ensure that the objects required for command execution can be context restored.

- g) If the TPM returns TPM\_RC\_SUCCESS, the object referenced by *objectHandle* will not be flushed and both *objectHandle* and *persistentHandle* may be used to access the object.

If *objectHandle* references a persistent object:

- h) The TPM shall return TPM\_RC\_RANGE if *objectHandle* is not in the proper range as determined by *auth*. If *auth* is TPM\_RC\_OWNER, *objectHandle* shall be in the inclusive range of 81 00 00 00<sub>16</sub> to 81 7F FF FF<sub>16</sub>. If *auth* is TPM\_RC\_PLATFORM, *objectHandle* may be any valid persistent object handle.
- i) If the TPM returns TPM\_RC\_SUCCESS, *objectHandle* will be removed from persistent memory and no longer be accessible.

NOTE 4            The persistent object is not converted to a transient object, as this would prevent the immediate revocation of an object by removing it from persistent memory.



30.5.2 Command and Response

Table 183 — TPM2\_EvictControl Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_EvictControl {NV}
TPMI_RH_PROVISION	@auth	TPM_RH_OWNER or TPM_RH_PLATFORM+{PP} Auth Handle: 1 Auth Role: USER
TPMI_DH_OBJECT	objectHandle	the handle of a loaded object Auth Index: None
TPMI_DH_PERSISTENT	persistentHandle	if <i>objectHandle</i> is a transient object handle, then this is the persistent handle for the object if <i>objectHandle</i> is a persistent object handle, then this shall be the same value as <i>persistentHandle</i>

Table 184 — TPM2\_EvictControl Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 30.5.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "EvictControl_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	an object with <i>temporary</i> , <i>stClear</i> or <i>publicOnly</i> attribute SET cannot be made persistent
TPM_RC_HIERARCHY	<i>auth</i> cannot authorize the operation in the hierarchy of <i>evictObject</i>
TPM_RC_HANDLE	<i>evictHandle</i> of the persistent object to be evicted is not the same as the <i>persistentHandle</i> argument
TPM_RC_NV_HANDLE	<i>persistentHandle</i> is unavailable
TPM_RC_NV_SPACE	no space in NV to make <i>evictHandle</i> persistent
TPM_RC_RANGE	<i>persistentHandle</i> is not in the range corresponding to the hierarchy of <i>evictObject</i>

```

3  TPM_RC
4  TPM2_EvictControl(
5      EvictControl_In      *in          // IN: input parameter list
6  )
7  {
8      TPM_RC      result;
9      OBJECT      *evictObject;
10
11     // The command needs NV update. Check if NV is available.
12     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
13     // this point
14     result = NvIsAvailable();
15     if(result != TPM_RC_SUCCESS) return result;
16
17     // Input Validation
18
19     // Get internal object pointer
20     evictObject = ObjectGet(in->objectHandle);
21
22     // Temporary, stClear or public only objects can not be made persistent
23     if( evictObject->attributes.temporary == SET
24         || evictObject->attributes.stClear == SET
25         || evictObject->attributes.publicOnly == SET
26     )
27         return TPM_RC_ATTRIBUTES + RC_EvictControl_objectHandle;
28
29     // If objectHandle refers to a persistent object, it should be the same as
30     // input persistentHandle
31     if( evictObject->attributes.evict == SET
32         && evictObject->evictHandle != in->persistentHandle
33     )
34         return TPM_RC_HANDLE + RC_EvictControl_objectHandle;
35
36     // Additional auth validation
37     if(in->auth == TPM_RH_PLATFORM)
38     {
39         // To make persistent
40         if(evictObject->attributes.evict == CLEAR)
41         {
42             // Platform auth can not set evict object in storage or endorsement
43             // hierarchy

```

```
44     if(evictObject->attributes.ppsHierarchy == CLEAR)
45         return TPM_RC_HIERARCHY + RC_EvictControl_objectHandle;
46
47     // Platform cannot use a handle outside of platform persistent range.
48     if(!NvIsPlatformPersistentHandle(in->persistentHandle))
49         return TPM_RC_RANGE + RC_EvictControl_persistentHandle;
50     }
51     // Platform auth can delete any persistent object
52 }
53 else if(in->auth == TPM_RH_OWNER)
54 {
55     // Owner auth can not set or clear evict object in platform hierarchy
56     if(evictObject->attributes.ppsHierarchy == SET)
57         return TPM_RC_HIERARCHY + RC_EvictControl_objectHandle;
58
59     // Owner cannot use a handle outside of owner persistent range.
60     if( evictObject->attributes.evict == CLEAR
61        && !NvIsOwnerPersistentHandle(in->persistentHandle)
62        )
63         return TPM_RC_RANGE + RC_EvictControl_persistentHandle;
64 }
65 else
66 {
67     // Other auth is not allowed in this command and should be filtered out
68     // at unmarshal process
69     pAssert(FALSE);
70 }
71
72 // Internal Data Update
73
74 // Change evict state
75 if(evictObject->attributes.evict == CLEAR)
76 {
77     // Make object persistent
78     // A TPM_RC_NV_HANDLE or TPM_RC_NV_SPACE error may be returned at this
79     // point
80     result = NvAddEvictObject(in->persistentHandle, evictObject);
81     if(result != TPM_RC_SUCCESS) return result;
82 }
83 else
84 {
85     // Delete the persistent object in NV
86     NvDeleteEntity(evictObject->evictHandle);
87 }
88
89 return TPM_RC_SUCCESS;
90
91 }
```

## 31 Clocks and Timers

### 31.1 TPM2\_ReadClock

#### 31.1.1 General Description

This command reads the current TPMS\_TIME\_INFO structure that contains the current setting of *Time*, *Clock*, *resetCount*, and *restartCount*.

No authorization sessions of any type are allowed with this command and tag is required to be TPM\_ST\_NO\_SESSIONS.

NOTE This command is intended to allow the TCB to have access to values that have the potential to be privacy sensitive. The values may be read without authorization because the TCB will not disclose these values. Since they are not signed and cannot be accessed in a command that uses an authorization session, it is not possible for any entity, other than the TCB, to be assured that the values are accurate.

## 31.1.2 Command and Response

Table 185 — TPM2\_ReadClock Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ReadClock

Table 186 — TPM2\_ReadClock Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	returnCode	
TPMS_TIME_INFO	currentTime	

### 31.1.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "ReadClock_fp.h"
3  TPM_RC
4  TPM2_ReadClock(
5      ReadClock_Out *out          // OUT: output parameter list
6  )
7  {
8  // Command Output
9
10     out->currentTime.time = g_time;
11     TimeFillInfo(&out->currentTime.clockInfo);
12
13     return TPM_RC_SUCCESS;
14 }
```

## 31.2 TPM2\_ClockSet

### 31.2.1 General Description

This command is used to advance the value of the TPM's *Clock*. The command will fail if *newTime* is less than the current value of *Clock* or if the new time is greater than FF FF 00 00 00 00 00 00<sub>16</sub>. If both of these checks succeed, *Clock* is set to *newTime*. If either of these checks fails, the TPM shall return TPM\_RC\_VALUE and make no change to *Clock*.

NOTE This maximum setting would prevent *Clock* from rolling over to zero for approximately 8,000 years if the *Clock* update rate was set so that TPM time was passing 33 percent faster than real time. This would still be more than 6,000 years before *Clock* would roll over to zero. Because *Clock* will not roll over in the lifetime of the TPM, there is no need for external software to deal with the possibility that *Clock* may wrap around.

If the value of *Clock* after the update makes the volatile and non-volatile versions of TPMS\_CLOCK\_INFO.*clock* differ by more than the reported update interval, then the TPM shall update the non-volatile version of TPMS\_CLOCK\_INFO.*clock* before returning.

This command requires *platformAuth* or *ownerAuth*.

## 31.2.2 Command and Response

Table 187 — TPM2\_ClockSet Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ClockSet {NV}
TPMI_RH_PROVISION	@auth	TPM_RH_OWNER or TPM_RH_PLATFORM+{PP} Auth Handle: 1 Auth Role: USER
UINT64	newTime	new <i>Clock</i> setting in milliseconds

Table 188 — TPM2\_ClockSet Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	returnCode	



## 31.2.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "ClockSet_fp.h"

```

Read the current TPMS\_TIMER\_INFO structure settings

Error Returns	Meaning
TPM_RC_VALUE	invalid new clock

```

3  TPM_RC
4  TPM2_ClockSet(
5      ClockSet_In *in                // IN: input parameter list
6  )
7  {
8      #define CLOCK_UPDATE_MASK  ((1ULL << NV_CLOCK_UPDATE_INTERVAL) - 1)
9      UINT64      clockNow;
10
11     // Input Validation
12
13     // new time can not be bigger than 0xFFFF000000000000 or smaller than
14     // current clock
15     if(in->newTime > 0xFFFF000000000000ULL
16         || in->newTime < go.clock)
17         return TPM_RC_VALUE + RC_ClockSet_newTime;
18
19     // Internal Data Update
20
21     // Internal Data Update
22     clockNow = go.clock;    // grab the old value
23     go.clock = in->newTime; // set the new value
24     // Check to see if the update has caused a need for an nvClock update
25     if((in->newTime & CLOCK_UPDATE_MASK) > (clockNow & CLOCK_UPDATE_MASK))
26     {
27         NvWriteReserved(NV_CLOCK, &go.clock);
28         // Now the time state is safe
29         go.clockSafe = YES;
30     }
31
32     return TPM_RC_SUCCESS;
33 }

```

### 31.3 TPM2\_ClockRateAdjust

#### 31.3.1 General Description

This command adjusts the rate of advance of *Clock* and *Time* to provide a better approximation to real time.

The *rateAdjust* value is relative to the current rate and not the nominal rate of advance.

EXAMPLE 1 If this command had been called three times with *rateAdjust* = TPM\_CLOCK\_COARSE\_SLOWER and once with *rateAdjust* = TPM\_CLOCK\_COARSE\_FASTER, the net effect will be as if the command had been called twice with *rateAdjust* = TPM\_CLOCK\_COARSE\_SLOWER.

The range of adjustment shall be sufficient to allow *Clock* and *Time* to advance at real time but no more. If the requested adjustment would make the rate advance faster or slower than the nominal accuracy of the input frequency, the TPM shall return TPM\_RC\_VALUE.

EXAMPLE 2 If the frequency tolerance of the TPM's input clock is +/-10 percent, then the TPM will return TPM\_RC\_VALUE if the adjustment would make *Clock* run more than 10 percent faster or slower than nominal. That is, if the input oscillator were nominally 100 megahertz (MHz), then 1 millisecond (ms) would normally take 100,000 counts. The update *Clock* should be adjustable so that 1 ms is between 90,000 and 110,000 counts.

The interpretation of “fine” and “coarse” adjustments is implementation-specific.

The nominal rate of advance for *Clock* and *Time* shall be accurate to within 15 percent. That is, with no adjustment applied, *Clock* and *Time* shall be advanced at a rate within 15 percent of actual time.

NOTE If the adjustments are incorrect, it will be possible to make the difference between advance of *Clock/Time* and real time to be as much as  $1.15^2$  or  $\sim 1.33$ .

Changes to the current *Clock* update rate adjustment need not be persisted across TPM power cycles.

31.3.2 Command and Response

Table 189 — TPM2\_ClockRateAdjust Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ClockRateAdjust
TPMI_RH_PROVISION	@auth	TPM_RH_OWNER or TPM_RH_PLATFORM+{PP} Auth Handle: 1 Auth Role: USER
TPM_CLOCK_ADJUST	rateAdjust	Adjustment to current <i>Clock</i> update rate

Table 190 — TPM2\_ClockRateAdjust Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	returnCode	

### 31.3.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "ClockRateAdjust_fp.h"
3  TPM_RC
4  TPM2_ClockRateAdjust(
5      ClockRateAdjust_In      *in           // IN: input parameter list
6  )
7  {
8  // Internal Data Update
9      TimeSetAdjustRate(in->rateAdjust);
10
11     return TPM_RC_SUCCESS;
12 }
```

## 32 Capability Commands

### 32.1 Introduction

The TPM has numerous values that indicate the state, capabilities, and properties of the TPM. These values are needed for proper management of the TPM. The TPM2\_GetCapability() command is used to access these values.

TPM2\_GetCapability() allows reporting of multiple values in a single call. The values are grouped according to type.

NOTE            TPM2\_TestParms() is used to determine if a TPM supports a particular combination of algorithm parameters

### 32.2 TPM2\_GetCapability

#### 32.2.1 General Description

This command returns various information regarding the TPM and its current state.

The *capability* parameter determines the category of data returned. The *property* parameter selects the first value of the selected category to be returned. If there is no property that corresponds to the value of *property*, the next higher value is returned, if it exists.

EXAMPLE 1        The list of handles of transient objects currently loaded in the TPM may be read one at a time. On the first read, set the property to TRANSIENT\_FIRST and *propertyCount* to one. If a transient object is present, the lowest numbered handle is returned and *moreData* will be YES if transient objects with higher handles are loaded. On the subsequent call, use returned handle value plus 1 in order to access the next higher handle.

The *propertyCount* parameter indicates the number of capabilities in the indicated group that are requested. The TPM will return the number of requested values (*propertyCount*) or until the last property of the requested type has been returned.

NOTE 1            The type of the capability is determined by a combination of *capability* and *property*.

When all of the properties of the requested type have been returned, the *moreData* parameter in the response will be set to NO. Otherwise, it will be set to YES.

NOTE 2            The *moreData* parameter will be YES if there are more properties even if the requested number of capabilities has been returned.

The TPM is not required to return more than one value at a time. It is not required to provide the same number of values in response to subsequent requests.

EXAMPLE 2        A TPM may return 4 properties in response to a TPM2\_GetCapability(*capability* = TPM\_CAP\_TPM\_PROPERTY, *property* = TPM\_PT\_MANUFACTURER, *propertyCount* = 8 ) and for a latter request with the same parameters, the TPM may return as few as one and as many as 8 values.

When the TPM is in Failure mode, a TPM is required to allow use of this command for access of the following capabilities:

- TPM\_PT\_MANUFACTURER
- TPM\_PT\_VENDOR\_STRING\_1
- TPM\_PT\_VENDOR\_STRING\_2<sup>(3)</sup>
- TPM\_PT\_VENDOR\_STRING\_3<sup>(3)</sup>
- TPM\_PT\_VENDOR\_STRING\_4<sup>(3)</sup>
- TPM\_PT\_VENDOR\_TPM\_TYPE
- TPM\_PT\_FIRMWARE\_VERSION\_1
- TPM\_PT\_FIRMWARE\_VERSION\_2

NOTE 3 If the vendor string does not require one of these values, the property type does not need to exist.

A vendor may optionally allow the TPM to return other values.

If in Failure mode and a capability is requested that is not available in Failure mode, the TPM shall return no value.

EXAMPLE 3 Assume the TPM is in Failure mode and the TPM only supports reporting of the minimum required set of properties (the limited set to TPML\_TAGGED\_PCR\_PROPERTY values). If a TPM2\_GetCapability is received requesting a capability that has a property type value greater than TPM\_PT\_FIRMWARE\_VERSION\_2, the TPM will return a zero length list with the moreData parameter set to NO. If the property type is less than TPM\_PT\_MANUFACTURER, the TPM will return TPM\_PT\_MANUFACTURER.

In Failure mode, *tag* is required to be TPM\_ST\_NO\_SESSIONS or the TPM shall return TPM\_RC\_FAILURE.

The capability categories and the types of the return values are:

<b>capability</b>	<b>property</b>	<b>Return Type</b>
TPM_CAP_ALGS	TPM_ALG_ID <sup>(1)</sup>	TPML_ALG_PROPERTY
TPM_CAP_HANDLES	TPM_HANDLE	TPML_HANDLE
TPM_CAP_COMMANDS	TPM_CC	TPML_CCA
TPM_CAP_PP_COMMANDS	TPM_CC	TPML_CC
TPM_CAP_AUDIT_COMMANDS	TPM_CC	TPML_CC
TPM_CAP_PCRS	Reserved	TPML_PCR_SELECTION
TPM_CAP_TPM_PROPERTIES	TPM_PT	TPML_TAGGED_TPM_PROPERTY
TPM_CAP_PCR_PROPERTIES	TPM_PT_PCR	TPML_TAGGED_PCR_PROPERTY
TPM_CAP_ECC_CURVE	TPM_ECC_CURVE <sup>(1)</sup>	TPML_ECC_CURVE
TPM_CAP_VENDOR_PROPERTY	manufacturer specific	manufacturer-specific values
NOTES: (1) The TPM_ALG_ID or TPM_ECC_CURVE is cast to a UINT32		

- **TPM\_CAP\_ALGS** – Returns a list of TPMS\_ALG\_PROPERTIES. Each entry is an algorithm ID and a set of properties of the algorithm.
- **TPM\_CAP\_HANDLES** – Returns a list of all of the handles within the handle range of the *property* parameter. The range of the returned handles is determined by the handle type (the most-significant octet (MSO) of the *property*). Any of the defined handle types is allowed

EXAMPLE 4      If the MSO of *property* is TPM\_HT\_NV\_INDEX, then the TPM will return a list of NV Index values.

EXAMPLE 5      If the MSO of *property* is TPM\_HT\_PCR, then the TPM will return a list of PCR.

- For this capability, use of TPM\_HT\_LOADED\_SESSION and TPM\_HT\_SAVED\_SESSION is allowed. Requesting handles with a handle type of TPM\_HT\_LOADED\_SESSION will return handles for loaded sessions. The returned handle values will have a handle type of either TPM\_HT\_HMAC\_SESSION or TPM\_HT\_POLICY\_SESSION. If saved sessions are requested, all returned values will have the TPM\_HT\_HMAC\_SESSION handle type because the TPM does not track the session type of saved sessions.

NOTE 2            TPM\_HT\_LOADED\_SESSION and TPM\_HT\_HMAC\_SESSION have the same value, as do TPM\_HT\_SAVED\_SESSION and TPM\_HT\_POLICY\_SESSION. It is not possible to request that the TPM return a list of loaded HMAC sessions without including the policy sessions.

- **TPM\_CAP\_COMMANDS** – Returns a list of the command attributes for all of the commands implemented in the TPM, starting with the TPM\_CC indicated by the *property* parameter. If vendor specific commands are implemented, the vendor-specific command attribute with the lowest *commandIndex*, is returned after the non-vendor-specific (base) command.

NOTE 4            The type of the property parameter is a TPM\_CC while the type of the returned list is TPML\_CCA.

- **TPM\_CAP\_PP\_COMMANDS** – Returns a list of all of the commands currently requiring Physical Presence for confirmation of platform authorization. The list will start with the TPM\_CC indicated by *property*.
- **TPM\_CAP\_AUDIT\_COMMANDS** – Returns a list of all of the commands currently set for command audit.
- **TPM\_CAP\_PCRS** – Returns the current allocation of PCR in a TPML\_PCR\_SELECTION. The *property* parameter shall be zero. The TPM will always respond to this command with the full PCR allocation and *moreData* will be NO.
- **TPM\_CAP\_TPM\_PROPERTIES** – Returns a list of tagged properties. The tag is a TPM\_PT and the property is a 32-bit value. The properties are returned in groups. Each property group is on a 256-value boundary (that is, the boundary occurs when the TPM\_PT is evenly divisible by 256). The TPM will only return values in the same group as the *property* parameter in the command.
- **TPM\_CAP\_PCR\_PROPERTIES** – Returns a list of tagged PCR properties. The tag is a TPM\_PT\_PCR and the property is a TPMS\_PCR\_SELECT.

The input command property is a TPM\_PT\_PCR (see Part 2 for PCR properties to be requested) that specifies the first property to be returned. If *propertyCount* is greater than 1, the list of properties begins with that property and proceeds in TPM\_PT\_PCR sequence.

NOTE 5            If the propertyCount selects an unimplemented property, the next higher implemented property is returned.

Each item in the list is a TPMS\_PCR\_SELECT structure that contains a bitmap of all PCR.

NOTE 6            A PCR index in all banks (all hash algorithms) has the same properties, so the hash algorithm is not specified here.

- TPM\_CAP\_TPM\_ECC\_CURVES – Returns a list of ECC curve identifiers currently available for use in the TPM.

The *moreData* parameter will have a value of YES if there are more values of the requested type that were not returned.

If no next capability exists, the TPM will return a zero-length list and *moreData* will have a value of NO.



## 32.2.2 Command and Response

Table 191 — TPM2\_GetCapability Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_GetCapability
TPM_CAP	capability	group selection; determines the format of the response
UINT32	property	further definition of information
UINT32	propertyCount	number of properties of the indicated type to return

Table 192 — TPM2\_GetCapability Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPMI_YES_NO	moreData	flag to indicate if there are more values of this type
TPMS_CAPABILITY_DATA	capabilityData	the capability data

## 32.2.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "GetCapability_fp.h"

```

Error Returns	Meaning
TPM_RC_HANDLE	value of <i>property</i> is in an unsupported handle range for the TPM_CAP_HANDLES <i>capability</i> value
TPM_RC_VALUE	invalid <i>capability</i> ; or <i>property</i> is not 0 for the TPM_CAP_PCERS <i>capability</i> value

```

3  TPM_RC
4  TPM2_GetCapability(
5      GetCapability_In    *in,                // IN: input parameter list
6      GetCapability_Out  *out               // OUT: output parameter list
7  )
8  {
9      // Command Output
10
11     // Set output capability type the same as input type
12     out->capabilityData.capability = in->capability;
13
14     switch(in->capability)
15     {
16     case TPM_CAP_ALGS:
17         out->moreData = AlgorithmCapGetImplemented((TPM_ALG_ID) in->property,
18             in->propertyCount, &out->capabilityData.data.algorithms);
19         break;
20     case TPM_CAP_HANDLES:
21         switch(HandleGetType((TPM_HANDLE) in->property))
22         {
23         case TPM_HT_TRANSIENT:
24             // Get list of handles of loaded transient objects
25             out->moreData = ObjectCapGetLoaded((TPM_HANDLE) in->property,
26                 in->propertyCount,
27                 &out->capabilityData.data.handles);
28             break;
29         case TPM_HT_PERSISTENT:
30             // Get list of handles of persistent objects
31             out->moreData = NvCapGetPersistent((TPM_HANDLE) in->property,
32                 in->propertyCount,
33                 &out->capabilityData.data.handles);
34             break;
35         case TPM_HT_NV_INDEX:
36             // Get list of defined NV index
37             out->moreData = NvCapGetIndex((TPM_HANDLE) in->property,
38                 in->propertyCount,
39                 &out->capabilityData.data.handles);
40             break;
41         case TPM_HT_LOADED_SESSION:
42             // Get list of handles of loaded sessions
43             out->moreData = SessionCapGetLoaded((TPM_HANDLE) in->property,
44                 in->propertyCount,
45                 &out->capabilityData.data.handles);
46             break;
47         case TPM_HT_ACTIVE_SESSION:
48             // Get list of handles of
49             out->moreData = SessionCapGetSaved((TPM_HANDLE) in->property,
50                 in->propertyCount,
51                 &out->capabilityData.data.handles);
52             break;

```

```

53     case TPM_HT_PCR:
54         // Get list of handles of PCR
55         out->moreData = PCRCapGetHandles((TPM_HANDLE) in->property,
56                                         in->propertyCount,
57                                         &out->capabilityData.data.handles);
58         break;
59     case TPM_HT_PERMANENT:
60         // Get list of permanent handles
61         out->moreData = PermanentCapGetHandles(
62             (TPM_HANDLE) in->property,
63             in->propertyCount,
64             &out->capabilityData.data.handles);
65         break;
66     default:
67         // Unsupported input handle type
68         return TPM_RC_HANDLE + RC_GetCapability_property;
69         break;
70     }
71     break;
72     case TPM_CAP_COMMANDS:
73         out->moreData = CommandCapGetCCList((TPM_CC) in->property,
74                                             in->propertyCount,
75                                             &out->capabilityData.data.command);
76         break;
77     case TPM_CAP_PP_COMMANDS:
78         out->moreData = PhysicalPresenceCapGetCCList((TPM_CC) in->property,
79                                                     in->propertyCount, &out->capabilityData.data.ppCommands);
80         break;
81     case TPM_CAP_AUDIT_COMMANDS:
82         out->moreData = CommandAuditCapGetCCList((TPM_CC) in->property,
83                                                  in->propertyCount,
84                                                  &out->capabilityData.data.auditCommands);
85         break;
86     case TPM_CAP_PCERS:
87         // Input property must be 0
88         if(in->property != 0)
89             return TPM_RC_VALUE + RC_GetCapability_property;
90         out->moreData = PCRCapGetAllocation(in->propertyCount,
91                                           &out->capabilityData.data.assignedPCR);
92         break;
93     case TPM_CAP_PCR_PROPERTIES:
94         out->moreData = PCRCapGetProperties((TPM_PT_PCR) in->property,
95                                           in->propertyCount,
96                                           &out->capabilityData.data.pcrProperties);
97         break;
98     case TPM_CAP_TPM_PROPERTIES:
99         out->moreData = TPMCapGetProperties((TPM_PT) in->property,
100                                           in->propertyCount,
101                                           &out->capabilityData.data.tpmProperties);
102         break;
103 #ifdef TPM_ALG_ECC
104     case TPM_CAP_ECC_CURVES:
105         out->moreData = CryptCapGetECCCurve((TPM_ECC_CURVE ) in->property,
106                                           in->propertyCount,
107                                           &out->capabilityData.data.eccCurves);
108         break;
109 #endif // TPM_ALG_ECC
110     case TPM_CAP_VENDOR_PROPERTY:
111         // vendor property is not implemented
112     default:
113         // Unexpected TPM_CAP value
114         return TPM_RC_VALUE;
115         break;
116 }

```

```
117  
118     return TPM_RC_SUCCESS;  
119 }
```

### 32.3 TPM2\_TestParms

#### 32.3.1 General Description

This command is used to check to see if specific combinations of algorithm parameters are supported.

The TPM will unmarshal the provided TPMT\_PUBLIC\_PARMS. If the parameters unmarshal correctly, then the TPM will return TPM\_RC\_SUCCESS, indicating that the parameters are valid for the TPM. The TPM will return the appropriate unmarshaling error if a parameter is not valid.

## 32.3.2 Command and Response

Table 193 — TPM2\_TestParms Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_TestParms
TPMT_PUBLIC_PARMS	parameters	algorithm parameters to be validated

Table 194 — TPM2\_TestParms Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	TPM_RC

### 32.3.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "TestParms_fp.h"
3  TPM_RC
4  TPM2_TestParms(
5      TestParms_In      *in          // IN: input parameter list
6  )
7  {
8      // Input parameter is not reference in command action
9      in = NULL;
10
11     // The parameters are tested at unmarshal process. We do nothing in command
12     // action
13     return TPM_RC_SUCCESS;
14 }
```

## 33 Non-volatile Storage

### 33.1 Introduction

The NV commands are used to create, update, read, and delete allocations of space in NV memory. Before an Index may be used, it must be defined (TPM2\_NV\_DefineSpace()).

An Index may be modified if the proper write authorization is provided or read if the proper read authorization is provided. Different controls are available for reading and writing.

An Index may have an Index-specific *authValue* and *authPolicy*. The *authValue* may be used to authorize reading if TPMA\_NV\_AUTHREAD is SET and writing if TPMA\_NV\_AUTHWRITE is SET. The *authPolicy* may be used to authorize reading if TPMA\_NV\_POLICYREAD is SET and writing if TPMA\_NV\_POLICYWRITE is SET.

TPMA\_NV\_PPREAD and TPMA\_NV\_PPWRITE indicate if reading or writing of the NV Index may be authorized by *platformAuth* or *platformPolicy*.

TPMA\_NV\_OWNERREAD and TPMA\_NV\_OWNERWRITE indicate if reading or writing of the NV Index may be authorized by *ownerAuth* or *ownerPolicy*.

If an operation on an NV index requires authorization, and the *authHandle* parameter is the handle of an NV Index, then the *nvIndex* parameter must have the same value or this TPM will return TPM\_RC\_NV\_AUTHORIZATION.

NOTE 1 This check ensures that the authorization that was provided is associated with the NV Index being authorized.

For creating an Index, *ownerAuth* may not be used if *shEnable* is CLEAR and *platformAuth* may not be used if *phEnable* is CLEAR.

If an Index was defined using *platformAuth*, then that Index is not accessible when *phEnable* is CLEAR. If an Index was defined using *ownerAuth*, then that Index is not accessible when *shEnable* is CLEAR.

For read access control, any combination of TPMA\_NV\_PPREAD, TPMA\_NV\_OWNERREAD, TPMA\_NV\_AUTHREAD, or TPMA\_NV\_POLICYREAD is allowed as long as at least one is SET.

For write access control, any combination of TPMA\_NV\_PPWRITE, TPMA\_NV\_OWNERWRITE, TPMA\_NV\_AUTHWRITE, or TPMA\_NV\_POLICYWRITE is allowed as long as at least one is SET.

If an Index has been defined and not written, then any operation on the NV Index that requires read authorization will fail (TPM\_RC\_NV\_INITIALIZED). This check may be made before or after other authorization checks but shall be performed before checking the NV Index *authValue*. An authorization failure due to the NV Index not having been written shall not be logged by the dictionary attack logic.

If TPMA\_NV\_CLEAR\_STCLEAR is SET, then the TPMA\_NV\_WRITTEN will be CLEAR on each TPM2\_Startup(TPM\_SU\_CLEAR). TPMA\_NV\_CLEAR\_STCLEAR shall not be SET if TPMA\_NV\_COUNTER is SET.

The code in the “Detailed Actions” clause of each command is written to interface with an implementation-dependent library that allows access to NV memory. The actions assume no specific layout of the structure of the NV data.

Only one NV Index may be directly referenced in a command.

NOTE 2 This means that, if *authHandle* references an NV Index, then *nvIndex* will have the same value. However, this does not limit the number of changes that may occur as side effects. For example, any number of NV Indexes might be relocated as a result of deleting or adding a NV Index.



### 33.2 NV Counters

When an Index has the TPMA\_NV\_COUNTER attribute set, it behaves as a monotonic counter and may only be updated using TPM2\_NV\_Increment().

When an NV counter is created, the TPM shall initialize the 8-octet counter value with a number that is greater than any count value for any NV counter on the TPM since the time of TPM manufacture.

An NV counter may be defined with the TPMA\_NV\_ORDERLY attribute to indicate that the NV Index is expected to be modified at a high frequency and that the data is only required to persist when the TPM goes through an orderly shutdown process. The TPM may update the counter value in RAM and occasionally update the non-volatile version of the counter. An orderly shutdown is one occasion to update the non-volatile count. If the difference between the volatile and non-volatile version of the counter becomes as large as MAX\_ORDERLY\_COUNT, this shall be another occasion for updating the non-volatile count.

Before an NV counter can be used, the TPM shall validate that the count is not less than a previously reported value. If the TPMA\_NV\_ORDERLY attribute is not SET, or if the TPM experienced an orderly shutdown, then the count is assumed to be correct. If the TPMA\_NV\_ORDERLY attribute is SET, and the TPM shutdown was not orderly, then the TPM shall OR MAX\_ORDERLY\_COUNT to the contents of the non-volatile counter and set that as the current count.

NOTE 1            Because the TPM would have updated the NV Index if the difference between the count values was equal to MAX\_ORDERLY\_COUNT + 1, the highest value that could have been in the NV Index is MAX\_ORDERLY\_COUNT so it is safe to restore that value.

NOTE 2            The TPM may implement the RAM portion of the counter such that the effective value of the NV counter is the sum of both the volatile and non-volatile parts. If so, then the TPM may initialize the RAM version of the counter to MAX\_ORDERLY\_COUNT and no update of NV is necessary.

NOTE 3            When a new NV counter is created, the TPM may search all the counters to determine which has the highest value. In this search, the TPM would use the sum of the non-volatile and RAM portions of the counter. The RAM portion of the counter shall be properly initialized to reflect shutdown process (orderly or not) of the TPM.

### 33.3 TPM2\_NV\_DefineSpace

#### 33.3.1 General Description

This command defines the attributes of an NV Index and causes the TPM to reserve space to hold the data associated with the NV Index. If a definition already exists at the NV Index, the TPM will return TPM\_RC\_NV\_DEFINED.

The TPM will return TPM\_RC\_ATTRIBUTES if more than one of TPMA\_NV\_COUNTER, TPMA\_NV\_BITS, or TPMA\_NV\_EXTEND is SET in *publicInfo*.

NOTE It is not required that any of these three attributes be set.

The TPM shall return TPM\_RC\_ATTRIBUTES if TPMA\_NV\_WRITTEN, TPM\_NV\_READLOCKED, or TPMA\_NV\_WRITELOCKED is SET.

If TPMA\_NV\_COUNTER or TPMA\_NV\_BITS is SET, then *publicInfo*→*dataSize* shall be set to eight (8) or the TPM shall return TPM\_RC\_SIZE.

If TPMA\_NV\_EXTEND is SET, then *publicInfo*→*dataSize* shall match the digest size of the *publicInfo.nameAlg* or the TPM shall return TPM\_RC\_SIZE.

If the NV Index is an ordinary Index and *publicInfo*→*dataSize* is larger than supported by the TPM implementation then the TPM shall return TPM\_RC\_SIZE.

NOTE The limit for the data size may vary according to the type of the index. For example, if the index is has TPMA\_NV\_ORDERLY SET, then the maximum size of an ordinary NV Index may be less than the size of an ordinary NV Index that has TPMA\_NV\_ORDERLY CLEAR.

At least one of TPMA\_NV\_PPREAD, TPMA\_NV\_OWNERREAD, TPMA\_NV\_AUTHREAD, or TPMA\_NV\_POLICYREAD shall be SET or the TPM shall return TPM\_RC\_ATTRIBUTES.

At least one of TPMA\_NV\_PPWRITE, TPMA\_NV\_OWNERWRITE, TPMA\_NV\_AUTHWRITE, or TPMA\_NV\_POLICYWRITE shall be SET or the TPM shall return TPM\_RC\_ATTRIBUTES.

If TPMA\_NV\_CLEAR\_STCLEAR is SET, then TPMA\_NV\_COUNTER shall be CLEAR or the TPM shall return TPM\_RC\_ATTRIBUTES.

If *platformAuth/platformPolicy* is used for authorization, then TPMA\_NV\_PLATFORMCREATE shall be SET in *publicInfo*. If *ownerAuth/ownerPolicy* is used for authorization, TPMA\_NV\_PLATFORMCREATE shall be CLEAR in *publicInfo*. If TPMA\_NV\_PLATFORMCREATE is not set correctly for the authorization, the TPM shall return TPM\_RC\_ATTRIBUTES.

If TPMA\_NV\_POLICY\_DELETE is SET, then the authorization shall be with *platformAuth* or the TPM shall return TPM\_RC\_ATTRIBUTES.

If the implementation does not support TPM2\_NV\_Increment(), the TPM shall return TPM\_RC\_ATTRIBUTES if TPMA\_NV\_COUNTER is SET.

If the implementation does not support TPM2\_NV\_SetBits(), the TPM shall return TPM\_RC\_ATTRIBUTES if TPMA\_NV\_BITS is SET.

If the implementation does not support TPM2\_NV\_Extend(), the TPM shall return TPM\_RC\_ATTRIBUTES if TPMA\_NV\_EXTEND is SET.

If the implementation does not support TPM2\_NV\_UndefineSpaceSpecial(), the TPM shall return TPM\_RC\_ATTRIBUTES if TPMA\_NV\_POLICY\_DELETE is SET.

After the successful completion of this command, the NV Index exists but TPMA\_NV\_WRITTEN will be CLEAR. Any access of the NV data will return TPM\_RC\_NV\_UNINITIALIZED.

In some implementations, an NV Index with the TPMA\_NV\_COUNTER attribute may require special TPM resources that provide higher endurance than regular NV. For those implementations, if this command fails because of lack of resources, the TPM will return TPM\_RC\_NV\_COUNTER.

The value of *auth* is saved in the created structure. The size of *auth* is limited to be no larger than the size of the digest produced by the NV Index's *nameAlg*.

## 33.3.2 Command and Response

Table 195 — TPM2\_NV\_DefineSpace Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_NV_DefineSpace {NV}
TPMI_RH_PROVISION	@authHandle	TPM_RH_OWNER or TPM_RH_PLATFORM+{PP} Auth Index: 1 Auth Role: USER
TPM2B_AUTH	auth	the authorization data
TPM2B_NV_PUBLIC	publicInfo	the public parameters of the NV area

Table 196 — TPM2\_NV\_DefineSpace Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 33.3.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "NV_DefineSpace_fp.h"

```

Error Returns	Meaning
TPM_RC_NV_ATTRIBUTES	attributes of the index are not consistent
TPM_RC_NV_DEFINED	index already exists
TPM_RC_HIERARCHY	index already exists and belongs to a disabled hierarchy
TPM_RC_NV_SPACE	Insufficient space for the index
TPM_RC_SIZE	'auth->size' or 'publicInfo->authPolicy.size' is larger than the digest size of 'publicInfo->nameAlg', or 'publicInfo->dataSize' is not consistent with 'publicInfo->attributes'.

```

3  TPM_RC
4  TPM2_NV_DefineSpace(
5      NV_DefineSpace_In      *in          // IN: input parameter list
6  )
7  {
8      TPM_RC      result;
9      TPMA_NV     attributes;
10     UINT16      nameSize;
11
12     nameSize = CryptGetHashDigestSize(in->publicInfo.t.nvPublic.nameAlg);
13
14     // Check if NV is available. NvIsAvailable may return TPM_RC_NV_UNAVAILABLE
15     // TPM_RC_NV_RATE or TPM_RC_SUCCESS.
16     result = NvIsAvailable();
17     if(result != TPM_RC_SUCCESS)
18         return result;
19
20     // Input Validation
21
22     attributes = in->publicInfo.t.nvPublic.attributes;
23
24     //TPMS_NV_PUBLIC validation.
25     // Counters and bit fields must have a size of 8
26     if ( (attributes.TPMA_NV_COUNTER == SET || attributes.TPMA_NV_BITS == SET)
27         && (in->publicInfo.t.nvPublic.dataSize != 8))
28         return TPM_RC_SIZE + RC_NV_DefineSpace_publicInfo;
29
30     // check that the authPolicy consistent with hash algorithm
31     if( in->publicInfo.t.nvPublic.authPolicy.t.size != 0
32        && in->publicInfo.t.nvPublic.authPolicy.t.size != nameSize)
33         return TPM_RC_SIZE + RC_NV_DefineSpace_publicInfo;
34
35     // make sure that the authValue is not too large
36     MemoryRemoveTrailingZeros(&in->auth);
37     if(in->auth.t.size > nameSize)
38         return TPM_RC_SIZE + RC_NV_DefineSpace_auth;
39
40
41     //TPMA_NV validation.
42     // Locks may not be SET and written cannot be SET
43     if( attributes.TPMA_NV_WRITTEN == SET
44        || attributes.TPMA_NV_WRITELOCKED == SET
45        || attributes.TPMA_NV_READLOCKED == SET)

```

```

46     return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
47
48 // There must be a way to read the index
49 if(  attributes.TPMA_NV_OWNERREAD == CLEAR
50     && attributes.TPMA_NV_PPREAD == CLEAR
51     && attributes.TPMA_NV_AUTHREAD == CLEAR
52     && attributes.TPMA_NV_POLICYREAD == CLEAR)
53     return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
54
55 // There must be a way to write the index unless it is a bit field
56 // (can set up the bit field so that it is only write with NV_TestAndSet
57 // and can only be deleted)
58 if(  attributes.TPMA_NV_OWNERWRITE == CLEAR
59     && attributes.TPMA_NV_PPWRITE == CLEAR
60     && attributes.TPMA_NV_AUTHWRITE == CLEAR
61     && attributes.TPMA_NV_POLICYWRITE == CLEAR
62     && attributes.TPMA_NV_BITS == CLEAR)
63     return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
64
65 // Make sure that no attribute is used that is not supported by the proper
66 // command
67 #if CC_NV_Increment == NO
68     if( attributes.TPMA_NV_COUNTER == SET)
69         return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
70 #endif
71 #if CC_NV_SetBits == NO
72     if( attributes.TPMA_NV_BITS == SET)
73         return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
74 #endif
75 #if CC_NV_Extend == NO
76     if( attributes.TPMA_NV_EXTEND == SET)
77         return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
78 #endif
79 #if CC_NV_UndefineSpaceSpecial == NO
80     if( attributes.TPMA_NV_POLICY_DELETE == SET)
81         return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
82 #endif
83
84 // Can be COUNTER or BITS or EXTEND but not more than one
85 if(  attributes.TPMA_NV_COUNTER == SET
86     && attributes.TPMA_NV_BITS == SET)
87     return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
88 if(  attributes.TPMA_NV_COUNTER == SET
89     && attributes.TPMA_NV_EXTEND == SET)
90     return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
91 if(  attributes.TPMA_NV_BITS == SET
92     && attributes.TPMA_NV_EXTEND == SET)
93     return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
94
95 // An index with TPMA_NV_CLEAR_STCLEAR can't be a counter
96 if(  attributes.TPMA_NV_CLEAR_STCLEAR == SET
97     && attributes.TPMA_NV_COUNTER == SET)
98     return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
99
100 // The index is allowed to have one of GLOBALLOCK or WRITEDEFINE SET
101 if(  attributes.TPMA_NV_GLOBALLOCK == SET
102     && attributes.TPMA_NV_WRITEDEFINE == SET)
103     return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
104
105 // Make sure that the creator of the index can delete the index
106 if( ( in->publicInfo.t.nvPublic.attributes.TPMA_NV_PLATFORMCREATE == SET
107     && in->authHandle == TPM_RH_OWNER
108     )
109     || ( in->publicInfo.t.nvPublic.attributes.TPMA_NV_PLATFORMCREATE == CLEAR

```

```
110         && in->authHandle == TPM_RH_PLATFORM
111     )
112 )
113     return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_authHandle;
114
115 // If TPMA_NV_POLICY_DELETE is SET, then the index must be defined by
116 // the platform
117 if(    in->publicInfo.t.nvPublic.attributes.TPMA_NV_POLICY_DELETE == SET
118     && TPM_RH_PLATFORM != in->authHandle
119 )
120     return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
121
122 // If the NV index is used as a PCR, the data size must match the digest
123 // size
124 if(    in->publicInfo.t.nvPublic.attributes.TPMA_NV_EXTEND == SET
125     && in->publicInfo.t.nvPublic.dataSize != nameSize
126 )
127     return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
128
129 // See if the index is already defined. Error returns from NvIsUndefinedIndex()
130 // are TPM_RC_NV_DEFINED or TPM_RC_HIERARCHY
131 result = NvIsUndefinedIndex(in->publicInfo.t.nvPublic.nvIndex);
132 if(result != TPM_RC_SUCCESS)
133     return RcSafeAddToResult(result, RC_NV_DefineSpace_publicInfo);
134
135 // Internal Data Update
136 // define the space. A TPM_RC_NV_SPACE error may be returned at this point
137 result = NvDefineIndex(&in->publicInfo.t.nvPublic, &in->auth);
138 if(result != TPM_RC_SUCCESS)
139     return result;
140
141 return TPM_RC_SUCCESS;
142
143 }
```

### 33.4 TPM2\_NV\_UndefineSpace

#### 33.4.1 General Description

This command removes an Index from the TPM.

If *nvIndex* is not defined, the TPM shall return TPM\_RC\_NV\_DEFINED.

If *nvIndex* references an Index that has its TPMA\_NV\_PLATFORMCREATE attribute SET, the TPM shall return TPM\_RC\_NV\_AUTHORITY unless *platformAuth* is provided.

NOTE            An Index with TPMA\_NV\_PLATFORMCREATE CLEAR may be deleted with *platformAuth*.



33.4.2 Command and Response

**Table 197 — TPM2\_NV\_UndefineSpace Command**

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_NV_UndefineSpace {NV}
TPMI_RH_PROVISION	@authHandle	TPM_RH_OWNER or TPM_RH_PLATFORM+{PP} Auth Index: 1 Auth Role: USER
TPMI_RH_NV_INDEX	nvIndex	the NV Index to remove from NV space Auth Index: None

**Table 198 — TPM2\_NV\_UndefineSpace Response**

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 33.4.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "NV_UndefineSpace_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	TPMA_NV_POLICY_DELETE is SET in the Index referenced by <i>nvIndex</i> so this command may not be used to delete this Index (see TPM2_NV_UndefineSpaceSpecial())
TPM_RC_NV_AUTHORIZATION	attempt to use <i>ownerAuth</i> to delete an index created by the platform

```

3  TPM_RC
4  TPM2_NV_UndefineSpace(
5      NV_UndefineSpace_In *in          // IN: input parameter list
6  )
7  {
8      TPM_RC          result;
9      NV_INDEX        nvIndex;
10
11     // The command needs NV update. Check if NV is available.
12     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
13     // this point
14     result = NvIsAvailable();
15     if(result != TPM_RC_SUCCESS) return result;
16
17     // Input Validation
18
19     // Get NV index info
20     NvGetIndexInfo(in->nvIndex, &nvIndex);
21
22     // This command can't be used to delete an index with TPMA_NV_POLICY_DELETE SET
23     if(SET == nvIndex.publicArea.attributes.TPMA_NV_POLICY_DELETE)
24         return TPM_RC_ATTRIBUTES + RC_NV_UndefineSpace_nvIndex;
25
26     // The owner may only delete an index that was defined with ownerAuth. The
27     // platform may delete an index that was created with either auth.
28     if( in->authHandle == TPM_RH_OWNER
29         && nvIndex.publicArea.attributes.TPMA_NV_PLATFORMCREATE == SET)
30         return TPM_RC_NV_AUTHORIZATION;
31
32     // Internal Data Update
33
34     // Call implementation dependent internal routine to delete NV index
35     NvDeleteEntity(in->nvIndex);
36
37     return TPM_RC_SUCCESS;
38 }

```

## 33.5 TPM2\_NV\_UndefineSpaceSpecial

### 33.5.1 General Description

This command allows removal of a platform-created NV Index that has TPMA\_NV\_POLICY\_DELETE SET.

This command requires that the policy of the NV Index be satisfied before the NV Index may be deleted. Because administrative role is required, the policy must contain a command that sets the policy command code to TPM\_CC\_NV\_UndefineSpaceSpecial.

If *nvIndex* is not defined, the TPM shall return TPM\_RC\_NV\_DEFINED.

If *nvIndex* references an Index that has its TPMA\_NV\_PLATFORMCREATE or TPMA\_NV\_POLICY\_DELETE attribute CLEAR, the TPM shall return TPM\_RC\_NV\_ATTRIBUTES.

NOTE An Index with TPMA\_NV\_PLATFORMCREATE CLEAR may be deleted with TPM2\_UndefineSpace().

## 33.5.2 Command and Response

Table 199 — TPM2\_NV\_UndefineSpaceSpecial Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_NV_UndefineSpaceSpecial {NV}
TPMI_RH_NV_INDEX	@nvIndex	Index to be deleted Auth Index: 1 Auth Role: ADMIN
TPMI_RH_PLATFORM	@platform	TPM_RH_PLATFORM + {PP} Auth Index: 2 Auth Role: USER

Table 200 — TPM2\_NV\_UndefineSpaceSpecial Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 33.5.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "NV_UndefineSpaceSpecial_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	TPMA_NV_POLICY_DELETE is not SET in the Index referenced by <i>nvIndex</i>

```

3  TPM_RC
4  TPM2_NV_UndefineSpaceSpecial(
5      NV_UndefineSpaceSpecial_In *in           // IN: input parameter list
6  )
7  {
8      TPM_RC      result;
9      NV_INDEX    nvIndex;
10
11     // The command needs NV update. Check if NV is available.
12     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
13     // this point
14     result = NvIsAvailable();
15     if(result != TPM_RC_SUCCESS)
16         return result;
17
18     // Input Validation
19
20     // Get NV index info
21     NvGetIndexInfo(in->nvIndex, &nvIndex);
22
23     // This operation only applies when the TPMA_NV_POLICY_DELETE attribute is SET
24     if(CLEAR == nvIndex.publicArea.attributes.TPMA_NV_POLICY_DELETE)
25         return TPM_RC_ATTRIBUTES + RC_NV_UndefineSpaceSpecial_nvIndex;
26
27     // Internal Data Update
28
29     // Call implementation dependent internal routine to delete NV index
30     NvDeleteEntity(in->nvIndex);
31
32     return TPM_RC_SUCCESS;
33 }

```

### 33.6 TPM2\_NV\_ReadPublic

#### 33.6.1 General Description

This command is used to read the public area and Name of an NV Index. The public area of an Index is not privacy-sensitive and no authorization is required to read this data.

33.6.2 Command and Response

Table 201 — TPM2\_NV\_ReadPublic Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_NV_ReadPublic
TPMI_RH_NV_INDEX	nvIndex	the NV Index Auth Index: None

Table 202 — TPM2\_NV\_ReadPublic Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_NV_PUBLIC	nvPublic	the public area of the NV Index
TPM2B_NAME	nvName	the Name of the <i>nvIndex</i>

### 33.6.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "NV_ReadPublic_fp.h"
3  TPM_RC
4  TPM2_NV_ReadPublic(
5      NV_ReadPublic_In      *in,           // IN: input parameter list
6      NV_ReadPublic_Out     *out          // OUT: output parameter list
7  )
8  {
9      NV_INDEX      nvIndex;
10
11  // Command Output
12
13  // Get NV index info
14  NvGetIndexInfo(in->nvIndex, &nvIndex);
15
16  // Copy data to output
17  out->nvPublic.t.nvPublic = nvIndex.publicArea;
18
19  // Compute NV name
20  out->nvName.t.size = NvGetName(in->nvIndex, out->nvName.t.name);
21
22  return TPM_RC_SUCCESS;
23 }
```



## 33.7 TPM2\_NV\_Write

### 33.7.1 General Description

This command writes a value to an area in NV memory that was previously defined by TPM2\_NV\_DefineSpace().

Proper authorizations are required for this command as determined by TPMA\_NV\_PPWRITE; TPMA\_NV\_OWNERWRITE; TPMA\_NV\_AUTHWRITE; and, if TPMA\_NV\_POLICY\_WRITE is SET, the *authPolicy* of the NV Index.

If the TPMA\_NV\_WRITELOCKED attribute of the NV Index is SET, then the TPM shall return TPM\_RC\_NV\_LOCKED.

NOTE 1 If authorization sessions are present, they are checked before checks to see if writes to the NV Index are locked.

If TPMA\_NV\_COUNTER, TPMA\_NV\_BITS or TPMA\_NV\_EXTEND of the NV Index is SET, then the TPM shall return TPM\_RC\_NV\_ATTRIBUTE.

If the size of the *data* parameter plus the *offset* parameter adds to a value that is greater than the size of the NV Index *data*, the TPM shall return TPM\_RC\_NV\_RANGE and not write any data to the NV Index.

If the TPMA\_NV\_WRITEALL attribute of the NV Index is SET, then the TPM shall return TPM\_RC\_NV\_RANGE if the size of the *data* parameter of the command is not the same as the *data* field of the NV Index.

If all checks succeed, the TPM will merge the *data.size* octets of *data.buffer* value into the *nvIndex→data* starting at *nvIndex→data[offset]*. If the NV memory is implemented with a technology that has endurance limitations, the TPM shall check that the merged data is different from the current contents of the NV Index and only perform a write to NV memory if they differ.

After successful completion of this command, TPMA\_NV\_WRITTEN for the NV Index will be SET.

NOTE 2 Once SET, TPMA\_NV\_WRITTEN remains SET until the NV Index is undefined or the NV Index is cleared.

## 33.7.2 Command and Response

Table 203 — TPM2\_NV\_Write Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_NV_Write {NV}
TPMI_RH_NV_AUTH	@authHandle	handle indicating the source of the authorization value Auth Index: 1 Auth Role: USER
TPMI_RH_NV_INDEX	nvIndex	the NV Index of the area to write Auth Index: None
TPM2B_MAX_NV_BUFFER	data	the data to write
UINT16	offset	the offset into the NV Area

Table 204 — TPM2\_NV\_Write Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 33.7.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "NV_Write_fp.h"
3  #include "NV_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	Index referenced by <i>nvIndex</i> has either TPMA_NV_BITS, TPMA_NV_COUNTER, or TPMA_NV_EVENT attribute SET
TPM_RC_NV_AUTHORIZATION	the authorization was valid but the authorizing entity ( <i>authHandle</i> ) is not allowed to write to the Index referenced by <i>nvIndex</i>
TPM_RC_NV_LOCKED	Index referenced by <i>nvIndex</i> is write locked
TPM_RC_NV_RANGE	if TPMA_NV_WRITEALL is SET then the write is not the size of the Index referenced by <i>nvIndex</i> ; otherwise, the write extends beyond the limits of the Index

```

4  TPM_RC
5  TPM2_NV_Write(
6      NV_Write_In          *in          // IN: input parameter list
7  )
8  {
9      NV_INDEX          nvIndex;
10     TPM_RC          result;
11
12     // Input Validation
13
14     // Get NV index info
15     NvGetIndexInfo(in->nvIndex, &nvIndex);
16
17     // common access checks. NvWriteAccessChecks() may return
18     // TPM_RC_NV_AUTHORIZATION or TPM_RC_NV_LOCKED
19     result = NvWriteAccessChecks(in->authHandle, in->nvIndex);
20     if(result != TPM_RC_SUCCESS)
21         return result;
22
23     // Bits index, extend index or counter index may not be updated by
24     // TPM2_NV_Write
25     if( nvIndex.publicArea.attributes.TPMA_NV_COUNTER == SET
26        || nvIndex.publicArea.attributes.TPMA_NV_BITS == SET
27        || nvIndex.publicArea.attributes.TPMA_NV_EXTEND == SET)
28         return TPM_RC_ATTRIBUTES;
29
30     // Too much data
31     if((in->data.t.size + in->offset) > nvIndex.publicArea.dataSize)
32         return TPM_RC_NV_RANGE;
33
34     // If this index requires a full sized write, make sure that input range is
35     // full sized
36     if( nvIndex.publicArea.attributes.TPMA_NV_WRITEALL == SET
37        && in->data.t.size < nvIndex.publicArea.dataSize)
38         return TPM_RC_NV_RANGE;
39
40     // Internal Data Update
41
42     // Perform the write. This called routine will SET the TPMA_NV_WRITTEN
43     // attribute if it has not already been SET. If NV isn't available, an error
44     // will be returned.
45     return NvWriteIndexData(in->nvIndex, &nvIndex, in->offset,

```

```
46         in->data.t.size, in->data.t.buffer);  
47  
48     }
```

## 33.8 TPM2\_NV\_Increment

### 33.8.1 General Description

This command is used to increment the value in an NV Index that has TPMA\_NV\_COUNTER SET. The data value of the NV Index is incremented by one.

NOTE 1           The NV Index counter is an unsigned value.

If TPMA\_NV\_COUNTER is not SET in the indicated NV Index, the TPM shall return TPM\_RC\_ATTRIBUTES.

If TPMA\_NV\_WRITELOCKED is SET, the TPM shall return TPM\_RC\_NV\_LOCKED.

If TPMA\_NV\_WRITTEN is CLEAR, it will be SET.

If TPMA\_NV\_ORDERLY is SET, and the difference between the volatile and non-volatile versions of this field is greater than MAX\_ORDERLY\_COUNT, then the non-volatile version of the counter is updated.

NOTE 2           If a TPM implements TPMA\_NV\_ORDERLY and an Index is defined with TPMA\_NV\_ORDERLY and TPMA\_NV\_COUNTER both SET, then in the Event of a non-orderly shutdown, the non-volatile value for the counter Index will be advanced by MAX\_ORDERLY\_COUNT at the next TPM2\_Startup().

NOTE 3           An allowed implementation would keep a counter value in NV and a resettable counter in RAM. The reported value of the NV Index would be the sum of the two values. When the RAM count increments past the maximum allowed value (MAX\_ORDERLY\_COUNT), the non-volatile version of the count is updated with the sum of the values and the RAM count is reset to zero.

## 33.8.2 Command and Response

Table 205 — TPM2\_NV\_Increment Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_NV_Increment {NV}
TPMI_RH_NV_AUTH	@authHandle	handle indicating the source of the authorization value Auth Index: 1 Auth Role: USER
TPMI_RH_NV_INDEX	nvIndex	the NV Index to increment Auth Index: None

Table 206 — TPM2\_NV\_Increment Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 33.8.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "NV_Increment_fp.h"
3  #include "NV_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	NV index is not a counter
TPM_RC_NV_AUTHORIZATION	authorization failure
TPM_RC_NV_LOCKED	Index is write locked

```

4  TPM_RC
5  TPM2_NV_Increment(
6      NV_Increment_In      *in          // IN: input parameter list
7  )
8  {
9      TPM_RC      result;
10     NV_INDEX     nvIndex;
11     UINT64      countValue;
12
13
14     // Input Validation
15
16     // Common access checks, a TPM_RC_NV_AUTHORIZATION or TPM_RC_NV_LOCKED
17     // error may be returned at this point
18     result = NvWriteAccessChecks(in->authHandle, in->nvIndex);
19     if(result != TPM_RC_SUCCESS)
20         return result;
21
22     // Get NV index info
23     NvGetIndexInfo(in->nvIndex, &nvIndex);
24
25     // Make sure that this is a counter
26     if(nvIndex.publicArea.attributes.TPMA_NV_COUNTER != SET)
27         return TPM_RC_ATTRIBUTES + RC_NV_Increment_nvIndex;
28
29     // Internal Data Update
30
31     // If counter index is not been written, initialize it
32     if(nvIndex.publicArea.attributes.TPMA_NV_WRITTEN == CLEAR)
33         countValue = NvInitialCounter();
34     else
35         // Read NV data in native format for TPM CPU.
36         NvGetIntIndexData(in->nvIndex, &nvIndex, &countValue);
37
38     // Do the increment
39     countValue++;
40
41     // If this is an orderly counter that just rolled over, need to be able to
42     // write to NV to proceed. This check is done here, because NvWriteIndexData()
43     // does not see if the update is for counter rollover.
44     if(
45         nvIndex.publicArea.attributes.TPMA_NV_ORDERLY == SET
46         && (countValue & MAX_ORDERLY_COUNT) == 0)
47     {
48         result = NvIsAvailable();
49         if(result != TPM_RC_SUCCESS)
50             return result;
51
52         // Need to force an NV update

```

```
52     g_updateNV = TRUE;
53 }
54
55 // Write NV data back. A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may
56 // be returned at this point. If necessary, this function will set the
57 // TPMA_NV_WRITTEN attribute
58 return NvWriteIndexData(in->nvIndex, &nvIndex, 0, 8, &countValue);
59
60 }
```



### 33.9 TPM2\_NV\_Extend

#### 33.9.1 General Description

This command extends a value to an area in NV memory that was previously defined by TPM2\_NV\_DefineSpace.

If TPMA\_NV\_EXTEND is not SET, then the TPM shall return TPM\_RC\_ATTRIBUTES.

Proper write authorizations are required for this command as determined by TPMA\_NV\_PPWRITE, TPMA\_NV\_OWNERWRITE, TPMA\_NV\_AUTHWRITE, and the *authPolicy* of the NV Index.

After successful completion of this command, TPMA\_NV\_WRITTEN for the NV Index will be SET.

NOTE 1           Once SET, TPMA\_NV\_WRITTEN remains SET until the NV Index is undefined or the NV Index is cleared.

If the TPMA\_NV\_WRITELOCKED attribute of the NV Index is SET, then the TPM shall return TPM\_RC\_NV\_LOCKED.

NOTE 2           If authorization sessions are present, they are checked before checks to see if writes to the NV Index are locked.

The *data.buffer* parameter may be larger than the defined size of the NV Index.

The Index will be updated by:

$$nvIndex \rightarrow data_{new} := H_{nameAlg}(nvIndex \rightarrow data_{old} || data.buffer) \quad (38)$$

where

$H_{nameAlg}()$	the hash algorithm indicated in <i>nvIndex</i> $\rightarrow$ <i>nameAlg</i>
<i>nvIndex</i> $\rightarrow$ <i>data</i>	the value of the data field in the NV Index
<i>data.buffer</i>	the data buffer of the command parameter

NOTE 3           If TPMA\_NV\_WRITTEN is CLEAR, then *nvIndex*  $\rightarrow$  *data* is a Zero Digest.

## 33.9.2 Command and Response

Table 207 — TPM2\_NV\_Extend Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_NV_Extend {NV}
TPMI_RH_NV_AUTH	@authHandle	handle indicating the source of the authorization value Auth Index: 1 Auth Role: USER
TPMI_RH_NV_INDEX	nvIndex	the NV Index to extend Auth Index: None
TPM2B_MAX_NV_BUFFER	data	the data to extend

Table 208 — TPM2\_NV\_Extend Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 33.9.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "NV_Extend_fp.h"
3  #include "NV_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	the TPMA_NV_EXTEND attribute is not SET in the Index referenced by <i>nvIndex</i>
TPM_RC_NV_AUTHORIZATION	the authorization was valid but the authorizing entity ( <i>authHandle</i> ) is not allowed to write to the Index referenced by <i>nvIndex</i>
TPM_RC_NV_LOCKED	the Index referenced by <i>nvIndex</i> is locked for writing

```

4  TPM_RC
5  TPM2_NV_Extend(
6      NV_Extend_In      *in          // IN: input parameter list
7  )
8  {
9      TPM_RC      result;
10     NV_INDEX     nvIndex;
11
12     TPM2B_DIGEST oldDigest;
13     TPM2B_DIGEST newDigest;
14     HASH_STATE   hashState;
15
16     // Input Validation
17
18     // Common access checks, NvWriteAccessCheck() may return TPM_RC_NV_AUTHORIZATION
19     // or TPM_RC_NV_LOCKED
20     result = NvWriteAccessChecks(in->authHandle, in->nvIndex);
21     if(result != TPM_RC_SUCCESS)
22         return result;
23
24     // Get NV index info
25     NvGetIndexInfo(in->nvIndex, &nvIndex);
26
27     // Make sure that this is an extend index
28     if(nvIndex.publicArea.attributes.TPMA_NV_EXTEND != SET)
29         return TPM_RC_ATTRIBUTES + RC_NV_Extend_nvIndex;
30
31     // If the Index is not-orderly, or if this is the first write, NV will
32     // need to be updated.
33     if( nvIndex.publicArea.attributes.TPMA_NV_ORDERLY == CLEAR
34        || nvIndex.publicArea.attributes.TPMA_NV_WRITTEN == CLEAR)
35     {
36         // Check if NV is available. NvIsAvailable may return TPM_RC_NV_UNAVAILABLE
37         // TPM_RC_NV_RATE or TPM_RC_SUCCESS.
38         result = NvIsAvailable();
39         if(result != TPM_RC_SUCCESS)
40             return result;
41     }
42
43     // Internal Data Update
44
45     // Perform the write.
46     oldDigest.t.size = CryptGetHashDigestSize(nvIndex.publicArea.nameAlg);
47     if(nvIndex.publicArea.attributes.TPMA_NV_WRITTEN == SET)
48     {
49         NvGetIndexData(in->nvIndex, &nvIndex, 0,

```

```
50         oldDigest.t.size, oldDigest.t.buffer);
51     }
52     else
53     {
54         MemorySet(oldDigest.t.buffer, 0, oldDigest.t.size);
55     }
56     // Start hash
57     newDigest.t.size = CryptStartHash(nvIndex.publicArea.nameAlg, &hashState);
58
59     // Adding old digest
60     CryptUpdateDigest2B(&hashState, &oldDigest.b);
61
62     // Adding new data
63     CryptUpdateDigest2B(&hashState, &in->data.b);
64
65     // Complete hash
66     CryptCompleteHash2B(&hashState, &newDigest.b);
67
68     // Write extended hash back.
69     // Note, this routine will SET the TPMA_NV_WRITTEN attribute if necessary
70     return NvWriteIndexData(in->nvIndex, &nvIndex, 0,
71         newDigest.t.size, newDigest.t.buffer);
72 }
```

### 33.10 TPM2\_NV\_SetBits

#### 33.10.1 General Description

This command is used to SET bits in an NV Index that was created as a bit field. Any number of bits from 0 to 64 may be SET. The contents of *data* are ORed with the current contents of the NV Index starting at *offset*. The checks on *data* and *offset* are the same as for TPM2\_NV\_Write.

If TPMA\_NV\_WRITTEN is not SET, then, for the purposes of this command, the NV Index is considered to contain all zero bits and *data* is OR with that value.

If TPMA\_NV\_BITS is not SET, then the TPM shall return TPM\_RC\_ATTRIBUTES.

After successful completion of this command, TPMA\_NV\_WRITTEN for the NV Index will be SET.

NOTE TPMA\_NV\_WRITTEN will be SET even if no bits were SET.

## 33.10.2 Command and Response

Table 209 — TPM2\_NV\_SetBits Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_NV_SetBits {NV}
TPMI_RH_NV_AUTH	@authHandle	handle indicating the source of the authorization value Auth Index: 1 Auth Role: USER
TPMI_RH_NV_INDEX	nvIndex	NV Index of the area in which the bit is to be set Auth Index: None
UINT64	bits	the data to OR with the current contents

Table 210 — TPM2\_NV\_SetBits Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 33.10.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "NV_SetBits_fp.h"
3  #include "NV_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	the TPMA_NV_BITS attribute is not SET in the Index referenced by <i>nvIndex</i>
TPM_RC_NV_AUTHORIZATION	the authorization was valid but the authorizing entity ( <i>authHandle</i> ) is not allowed to write to the Index referenced by <i>nvIndex</i>
TPM_RC_NV_LOCKED	the Index referenced by <i>nvIndex</i> is locked for writing

```

4  TPM_RC
5  TPM2_NV_SetBits(
6      NV_SetBits_In          *in          // IN: input parameter list
7  )
8  {
9      TPM_RC          result;
10     NV_INDEX         nvIndex;
11     UINT64          bitValue;
12
13
14
15     // Input Validation
16
17     // Common access checks, NvWriteAccessCheck() may return TPM_RC_NV_AUTHORIZATION
18     // or TPM_RC_NV_LOCKED
19     // error may be returned at this point
20     result = NvWriteAccessChecks(in->authHandle, in->nvIndex);
21     if(result != TPM_RC_SUCCESS)
22         return result;
23
24     // Get NV index info
25     NvGetIndexInfo(in->nvIndex, &nvIndex);
26
27     // Make sure that this is a bit field
28     if(nvIndex.publicArea.attributes.TPMA_NV_BITS != SET)
29         return TPM_RC_ATTRIBUTES + RC_NV_SetBits_nvIndex;
30
31     // If the Index is not-orderly, or if this is the first write, NV will
32     // need to be updated.
33     if(    nvIndex.publicArea.attributes.TPMA_NV_ORDERLY == CLEAR
34         || nvIndex.publicArea.attributes.TPMA_NV_WRITTEN == CLEAR)
35     {
36         // Check if NV is available. NvIsAvailable may return TPM_RC_NV_UNAVAILABLE
37         // TPM_RC_NV_RATE or TPM_RC_SUCCESS.
38         result = NvIsAvailable();
39         if(result != TPM_RC_SUCCESS)
40             return result;
41     }
42
43     // Internal Data Update
44
45     // If index is not been written, initialize it
46     if(nvIndex.publicArea.attributes.TPMA_NV_WRITTEN == CLEAR)
47         bitValue = 0;
48     else
49         // Read index data

```

```
50     NvGetIntIndexData(in->nvIndex, &nvIndex, &bitValue);
51
52     // OR in the new bit setting
53     bitValue |= in->bits;
54
55     // Write index data back. If necessary, this function will SET
56     // TPMA_NV_WRITTEN.
57     return NvWriteIndexData(in->nvIndex, &nvIndex, 0, 8, &bitValue);
58
59 }
```



### 33.11 TPM2\_NV\_WriteLock

#### 33.11.1 General Description

If the TPMA\_NV\_WRITEDEFINE or TPMA\_NV\_WRITE\_STCLEAR attributes of an NV location are SET, then this command may be used to inhibit further writes of the NV Index.

Proper write authorization is required for this command as determined by TPMA\_NV\_PPWRITE, TPMA\_NV\_OWNERWRITE, TPMA\_NV\_AUTHWRITE, and the *authPolicy* of the NV Index.

It is not an error if TPMA\_NV\_WRITELOCKED for the NV Index is already SET.

If neither TPMA\_NV\_WRITEDEFINE nor TPMA\_NV\_WRITE\_STCLEAR of the NV Index is SET, then the TPM shall return TPM\_RC\_ATTRIBUTES.

If the command is properly authorized and TPMA\_NV\_WRITE\_STCLEAR or TPMA\_NV\_WRITEDEFINE is SET, then the TPM shall SET TPMA\_NV\_WRITELOCKED for the NV Index. TPMA\_NV\_WRITELOCKED will be clear on the next TPM2\_Startup(TPM\_SU\_CLEAR) unless TPMA\_NV\_WRITEDEFINE is SET.

## 33.11.2 Command and Response

Table 211 — TPM2\_NV\_WriteLock Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_NV_WriteLock {NV}
TPMI_RH_NV_AUTH	@authHandle	handle indicating the source of the authorization value Auth Index: 1 Auth Role: USER
TPMI_RH_NV_INDEX	nvIndex	the NV Index of the area to lock Auth Index: None

Table 212 — TPM2\_NV\_WriteLock Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 33.11.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "NV_WriteLock_fp.h"
3  #include "NV_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	neither TPMA_NV_WRITEDEFINE nor TPMA_NV_WRITE_STCLEAR is SET in Index referenced by <i>nvIndex</i>
TPM_RC_NV_AUTHORIZATION	the authorization was valid but the authorizing entity ( <i>authHandle</i> ) is not allowed to write to the Index referenced by <i>nvIndex</i>

```

4  TPM_RC
5  TPM2_NV_WriteLock(
6      NV_WriteLock_In *in          // IN: input parameter list
7  )
8  {
9      TPM_RC      result;
10     NV_INDEX     nvIndex;
11
12     // The command needs NV update. Check if NV is available.
13     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
14     // this point
15     result = NvIsAvailable();
16     if(result != TPM_RC_SUCCESS)
17         return result;
18
19     // Input Validation:
20
21     // Common write access checks, a TPM_RC_NV_AUTHORIZATION or TPM_RC_NV_LOCKED
22     // error may be returned at this point
23     result = NvWriteAccessChecks(in->authHandle, in->nvIndex);
24     if(result != TPM_RC_SUCCESS)
25     {
26         if(result == TPM_RC_NV_AUTHORIZATION)
27             return TPM_RC_NV_AUTHORIZATION;
28         // If write access failed because the index is already locked, then it is
29         // no error.
30         return TPM_RC_SUCCESS;
31     }
32
33
34     // Get NV index info
35     NvGetIndexInfo(in->nvIndex, &nvIndex);
36
37     // if non of TPMA_NV_WRITEDEFINE or TPMA_NV_WRITE_STCLEAR is set, the index
38     // can not be write-locked
39     if( nvIndex.publicArea.attributes.TPMA_NV_WRITEDEFINE == CLEAR
40         && nvIndex.publicArea.attributes.TPMA_NV_WRITE_STCLEAR == CLEAR)
41         return TPM_RC_ATTRIBUTES + RC_NV_WriteLock_nvIndex;
42
43     // Internal Data Update
44
45     // Set the WRITELOCK attribute
46     nvIndex.publicArea.attributes.TPMA_NV_WRITELOCKED = SET;
47
48     // Write index info back
49     NvWriteIndexInfo(in->nvIndex, &nvIndex);
50

```

```
51     return TPM_RC_SUCCESS;  
52 }
```

## 33.12 TPM2\_NV\_GlobalWriteLock

### 33.12.1 General Description

The command will SET TPMA\_NV\_WRITELOCKED for all indexes that have their TPMA\_NV\_GLOBALLOCK attribute SET.

If an Index has both TPMA\_NV\_WRITELOCKED and TPMA\_NV\_WRITEDEFINE SET, then this command will permanently lock the NV Index for writing.

**NOTE** If an Index is defined with TPMA\_NV\_GLOBALLOCK SET, then the global lock does not apply until the next time this command is executed.

This command requires either platformAuth/platformPolicy or ownerAuth/ownerPolicy.

## 33.12.2 Command and Response

Table 213 — TPM2\_NV\_GlobalWriteLock Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_NV_GlobalWriteLock
TPMI_RH_PROVISION	@authHandle	TPM_RH_OWNER or TPM_RH_PLATFORM+{PP} Auth Index: 1 Auth Role: USER

Table 214 — TPM2\_NV\_GlobalWriteLock Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

### 33.12.3 Detailed Actions

```
1  #include "InternalRoutines.h"
2  #include "NV_GlobalWriteLock_fp.h"
3  TPM_RC
4  TPM2_NV_GlobalWriteLock(
5      NV_GlobalWriteLock_In *in          // IN: input parameter list
6  )
7  {
8      TPM_RC          result;
9
10     // Input parameter is not reference in command action
11     in = NULL; // to silence compiler warnings.
12
13     // The command needs NV update. Check if NV is available.
14     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
15     // this point
16     result = NvIsAvailable();
17     if(result != TPM_RC_SUCCESS)
18         return result;
19
20     // Internal Data Update
21
22     // Implementation dependent method of setting the global lock
23     NvSetGlobalLock();
24
25     return TPM_RC_SUCCESS;
26 }
```

### 33.13 TPM2\_NV\_Read

#### 33.13.1 General Description

This command reads a value from an area in NV memory previously defined by TPM2\_NV\_DefineSpace().

Proper authorizations are required for this command as determined by TPMA\_NV\_PPREAD, TPMA\_NV\_OWNERREAD, TPMA\_NV\_AUTHREAD, and the *authPolicy* of the NV Index.

If TPMA\_NV\_READLOCKED of the NV Index is SET, then the TPM shall return TPM\_RC\_NV\_LOCKED.

NOTE If authorization sessions are present, they are checked before the read-lock status of the NV Index is checked.

If the *size* parameter plus the *offset* parameter adds to a value that is greater than the size of the NV Index *data* area, the TPM shall return TPM\_RC\_NV\_RANGE and not read any data from the NV Index.

If the NV Index has been defined but the TPMA\_NV\_WRITTEN attribute is CLEAR, then this command shall return TPM\_RC\_NV\_UNINITIALIZED even if *size* is zero.

The *data* parameter in the response may be encrypted using parameter encryption.



33.13.2 Command and Response

Table 215 — TPM2\_NV\_Read Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_NV_Read
TPMI_RH_NV_AUTH	@authHandle	the handle indicating the source of the authorization value Auth Index: 1 Auth Role: USER
TPMI_RH_NV_INDEX	nvIndex	the NV Index to be read Auth Index: None
UINT16	size	number of octets to read
UINT16	offset	octet offset into the area This value shall be less than or equal to the size of the <i>nvIndex</i> data.

Table 216 — TPM2\_NV\_Read Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_MAX_NV_BUFFER	data	the data read

## 33.13.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "NV_Read_fp.h"
3  #include "NV_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_NV_AUTHORIZATION	the authorization was valid but the authorizing entity ( <i>authHandle</i> ) is not allowed to read from the Index referenced by <i>nvIndex</i>
TPM_RC_NV_LOCKED	the Index referenced by <i>nvIndex</i> is read locked
TPM_RC_NV_RANGE	read range defined by <i>size</i> and <i>offset</i> is outside the range of the Index referenced by <i>nvIndex</i>
TPM_RC_NV_UNINITIALIZED	the Index referenced by <i>nvIndex</i> has not been initialized (written)

```

4  TPM_RC
5  TPM2_NV_Read(
6      NV_Read_In      *in,          // IN: input parameter list
7      NV_Read_Out     *out         // OUT: output parameter list
8  )
9  {
10     NV_INDEX         nvIndex;
11     TPM_RC           result;
12
13     // Input Validation
14
15     // Get NV index info
16     NvGetIndexInfo(in->nvIndex, &nvIndex);
17
18     // Common read access checks. NvReadAccessChecks() returns
19     // TPM_RC_NV_AUTHORIZATION, TPM_RC_NV_LOCKED, or TPM_RC_NV_UNINITIALIZED
20     // error may be returned at this point
21     result = NvReadAccessChecks(in->authHandle, in->nvIndex);
22     if(result != TPM_RC_SUCCESS)
23         return result;
24
25     // Too much data
26     if((in->size + in->offset) > nvIndex.publicArea.dataSize)
27         return TPM_RC_NV_RANGE;
28
29     // Command Output
30
31     // Set the return size
32     out->data.t.size = in->size;
33     // Perform the read
34     NvGetIndexData(in->nvIndex, &nvIndex, in->offset, in->size, out->data.t.buffer);
35
36     return TPM_RC_SUCCESS;
37 }

```

### 33.14 TPM2\_NV\_ReadLock

#### 33.14.1 General Description

If TPMA\_NV\_READ\_STCLEAR is SET in an Index, then this command may be used to prevent further reads of the NV Index until the next TPM2\_Startup (TPM\_SU\_CLEAR).

Proper authorizations are required for this command as determined by TPMA\_NV\_PPREAD, TPMA\_NV\_OWNERREAD, TPMA\_NV\_AUTHREAD, and the *authPolicy* of the NV Index.

NOTE Only an entity that may read an Index is allowed to lock the NV Index for read.

If the command is properly authorized and TPMA\_NV\_READ\_STCLEAR of the NV Index is SET, then the TPM shall SET TPMA\_NV\_READLOCKED for the NV Index. If TPMA\_NV\_READ\_STCLEAR of the NV Index is CLEAR, then the TPM shall return TPM\_RC\_NV\_ATTRIBUTE. TPMA\_NV\_READLOCKED will be CLEAR by the next TPM2\_Startup(TPM\_SU\_CLEAR).

It is not an error to use this command for an Index that is already locked for reading.

An Index that had not been written may be locked for reading.

## 33.14.2 Command and Response

Table 217 — TPM2\_NV\_ReadLock Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_NV_ReadLock
TPMI_RH_NV_AUTH	@authHandle	the handle indicating the source of the authorization value Auth Index: 1 Auth Role: USER
TPMI_RH_NV_INDEX	nvIndex	the NV Index to be locked Auth Index: None

Table 218 — TPM2\_NV\_ReadLock Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 33.14.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "NV_ReadLock_fp.h"
3  #include "NV_spt_fp.h"

```

Error Returns	Meaning
TPM_RC_ATTRIBUTES	TPMA_NV_READ_STCLEAR is not SET so Index referenced by <i>nvIndex</i> may not be write locked
TPM_RC_NV_AUTHORIZATION	the authorization was valid but the authorizing entity ( <i>authHandle</i> ) is not allowed to read from the Index referenced by <i>nvIndex</i>

```

4  TPM_RC
5  TPM2_NV_ReadLock(
6      NV_ReadLock_In *in          // IN: input parameter list
7  )
8  {
9      TPM_RC      result;
10     NV_INDEX     nvIndex;
11
12     // The command needs NV update. Check if NV is available.
13     // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
14     // this point
15     result = NvIsAvailable();
16     if(result != TPM_RC_SUCCESS) return result;
17
18     // Input Validation
19
20     // Common read access checks. NvReadAccessChecks() returns
21     // TPM_RC_NV_AUTHORIZATION, TPM_RC_NV_LOCKED, or TPM_RC_NV_UNINITIALIZED
22     // error may be returned at this point
23     result = NvReadAccessChecks(in->authHandle, in->nvIndex);
24     if(result != TPM_RC_SUCCESS)
25     {
26         if(result == TPM_RC_NV_AUTHORIZATION)
27             return TPM_RC_NV_AUTHORIZATION;
28         // Index is already locked for write
29         else if(result == TPM_RC_NV_LOCKED)
30             return TPM_RC_SUCCESS;
31
32         // If NvReadAccessChecks return TPM_RC_NV_UNINITIALIZED, then continue.
33         // It is not an error to read lock an uninitialized Index.
34     }
35
36     // Get NV index info
37     NvGetIndexInfo(in->nvIndex, &nvIndex);
38
39     // if TPMA_NV_READ_STCLEAR is not set, the index can not be read-locked
40     if(nvIndex.publicArea.attributes.TPMA_NV_READ_STCLEAR == CLEAR)
41         return TPM_RC_ATTRIBUTES + RC_NV_ReadLock_nvIndex;
42
43     // Internal Data Update
44
45     // Set the READLOCK attribute
46     nvIndex.publicArea.attributes.TPMA_NV_READLOCKED = SET;
47     // Write NV info back
48     NvWriteIndexInfo(in->nvIndex, &nvIndex);
49
50     return TPM_RC_SUCCESS;
51 }

```

### 33.15 TPM2\_NV\_ChangeAuth

#### 33.15.1 General Description

This command allows the authorization secret for an NV Index to be changed.

If successful, the authorization secret (*authValue*) of the NV Index associated with *nvIndex* is changed.

This command requires that a policy session be used for authorization of *nvIndex* so that the ADMIN role may be asserted and that *commandCode* in the policy session context shall be TPM\_CC\_NV\_ChangeAuth. That is, the policy must contain a specific authorization for changing the authorization value of the referenced object.

NOTE            The reason for this restriction is to ensure that the administrative actions on *nvIndex* require explicit approval while other commands may use policy that is not command-dependent.

The size of the *newAuth* value may be no larger than the size of authorization indicated when the NV Index was defined.

33.15.2 Command and Response

Table 219 — TPM2\_NV\_ChangeAuth Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_NV_ChangeAuth {NV}
TPMI_RH_NV_INDEX	@nvIndex	handle of the object Auth Index: 1 Auth Role: ADMIN
TPM2B_AUTH	newAuth	new authorization secret

Table 220 — TPM2\_NV\_ChangeAuth Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	

## 33.15.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "NV_ChangeAuth_fp.h"

```

Error Returns	Meaning
TPM_RC_SIZE	<i>newAuth</i> size is larger than the digest size of the Name algorithm for the Index referenced by <i>nvIndex</i>

```

3  TPM_RC
4  TPM2_NV_ChangeAuth(
5      NV_ChangeAuth_In      *in          // IN: input parameter list
6  )
7  {
8      TPM_RC      result;
9      NV_INDEX    nvIndex;
10
11     // Input Validation
12     // Check if NV is available. NvIsAvailable may return TPM_RC_NV_UNAVAILABLE
13     // TPM_RC_NV_RATE or TPM_RC_SUCCESS.
14     result = NvIsAvailable();
15     if(result != TPM_RC_SUCCESS) return result;
16
17     // Read index info from NV
18     NvGetIndexInfo(in->nvIndex, &nvIndex);
19
20     // Remove any trailing zeros that might have been added by the caller
21     // to obfuscate the size.
22     MemoryRemoveTrailingZeros(&(in->newAuth));
23
24     // Make sure that the authValue is no larger than the nameAlg of the Index
25     if(in->newAuth.t.size > CryptGetHashDigestSize(nvIndex.publicArea.nameAlg))
26         return TPM_RC_SIZE + RC_NV_ChangeAuth_newAuth;
27
28     // Internal Data Update
29     // Change auth
30     nvIndex.authValue = in->newAuth;
31     // Write index info back to NV
32     NvWriteIndexInfo(in->nvIndex, &nvIndex);
33
34     return TPM_RC_SUCCESS;
35 }

```



### 33.16 TPM2\_NV\_Certify

#### 33.16.1 General Description

The purpose of this command is to certify the contents of an NV Index or portion of an NV Index.

If proper authorization for reading the NV Index is provided, the portion of the NV Index selected by *size* and *offset* are included in an attestation block and signed using the key indicated by *signHandle*. The attestation also includes *size* and *offset* so that the range of the data can be determined.

NOTE See 20.1 for description of how the signing scheme is selected.

## 33.16.2 Command and Response

Table 221 — TPM2\_NV\_Certify Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_NV_Certify
TPMI_DH_OBJECT+	@signHandle	handle of the key used to sign the attestation structure Auth Index: 1 Auth Role: USER
TPMI_RH_NV_AUTH	@authHandle	handle indicating the source of the authorization value for the NV Index Auth Index: 2 Auth Role: USER
TPMI_RH_NV_INDEX	nvIndex	Index for the area to be certified Auth Index: None
TPM2B_DATA	qualifyingData	user-provided qualifying data
TPMT_SIG_SCHEME+	inScheme	signing scheme to use if the <i>scheme</i> for <i>signHandle</i> is TPM_ALG_NULL
UINT16	size	number of octets to certify
UINT16	offset	octet offset into the area This value shall be less than or equal to the size of the <i>nvIndex</i> data.

Table 222 — TPM2\_NV\_Certify Response

Type	Name	Description
TPM_ST	tag	see clause 8
UINT32	responseSize	
TPM_RC	responseCode	.
TPM2B_ATTEST	certifyInfo	the structure that was signed
TPMT_SIGNATURE	signature	the asymmetric signature over <i>certifyInfo</i> using the key referenced by <i>signHandle</i>

## 33.16.3 Detailed Actions

```

1  #include "InternalRoutines.h"
2  #include "Attest_spt_fp.h"
3  #include "NV_spt_fp.h"
4  #include "NV_Certify_fp.h"

```

Error Returns	Meaning
TPM_RC_NV_AUTHORIZATION	the authorization was valid but the authorizing entity ( <i>authHandle</i> ) is not allowed to read from the Index referenced by <i>nvIndex</i>
TPM_RC_KEY	<i>signHandle</i> does not reference a signing key
TPM_RC_NV_LOCKED	Index referenced by <i>nvIndex</i> is locked for reading
TPM_RC_NV_RANGE	<i>offset</i> plus <i>size</i> extends outside of the data range of the Index referenced by <i>nvIndex</i>
TPM_RC_NV_UNINITIALIZED	Index referenced by <i>nvIndex</i> has not been written
TPM_RC_SCHEME	<i>inScheme</i> is not an allowed value for the key definition

```

5  TPM_RC
6  TPM2_NV_Certify(
7      NV_Certify_In      *in,           // IN: input parameter list
8      NV_Certify_Out     *out          // OUT: output parameter list
9  )
10 {
11     TPM_RC              result;
12     NV_INDEX            nvIndex;
13     TPMS_ATTEST         certifyInfo;
14
15     // Attestation command may cause the orderlyState to be cleared due to
16     // the reporting of clock info.  If this is the case, check if NV is
17     // available first
18     if(gp.orderlyState != SHUTDOWN_NONE)
19     {
20         // The command needs NV update.  Check if NV is available.
21         // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
22         // this point
23         result = NvIsAvailable();
24         if(result != TPM_RC_SUCCESS)
25             return result;
26     }
27
28     // Input Validation
29
30     // Get NV index info
31     NvGetIndexInfo(in->nvIndex, &nvIndex);
32
33     // Common access checks.  A TPM_RC_NV_AUTHORIZATION or TPM_RC_NV_LOCKED
34     // error may be returned at this point
35     result = NvReadAccessChecks(in->authHandle, in->nvIndex);
36     if(result != TPM_RC_SUCCESS)
37         return result;
38
39     // See if the range to be certified is out of the bounds of the defined
40     // Index
41     if((in->size + in->offset) > nvIndex.publicArea.dataSize)
42         return TPM_RC_NV_RANGE;
43
44     // Command Output

```

```

45
46 // Filling in attest information
47 // Common fields
48 // FillInAttestInfo can return TPM_RC_SCHEME or TPM_RC_KEY
49 result = FillInAttestInfo(in->signHandle,
50                          &in->inScheme,
51                          &in->qualifyingData,
52                          &certifyInfo);
53 if(result != TPM_RC_SUCCESS)
54 {
55     if(result == TPM_RC_KEY)
56         return TPM_RC_KEY + RC_NV_Certify_signHandle;
57     else
58         return RcSafeAddToResult(result, RC_NV_Certify_inScheme);
59 }
60 // NV certify specific fields
61 // Attestation type
62 certifyInfo.type = TPM_ST_ATTEST_NV;
63
64 // Get the name of the index
65 certifyInfo.attested.nv.indexName.t.size =
66     NvGetName(in->nvIndex, certifyInfo.attested.nv.indexName.t.name);
67
68 // Set the return size
69 certifyInfo.attested.nv.nvContents.t.size = in->size;
70
71 // Set the offset
72 certifyInfo.attested.nv.offset = in->offset;
73
74 // Perform the read
75 NvGetIndexData(in->nvIndex, &nvIndex,
76               in->offset, in->size,
77               certifyInfo.attested.nv.nvContents.t.buffer);
78
79 // Sign attestation structure. A NULL signature will be returned if
80 // signHandle is TPM_RH_NULL. SignAttestInfo() may return TPM_RC_VALUE,
81 // TPM_RC_SCHEME or TPM_RC_ATTRIBUTES.
82 // Note: SignAttestInfo may return TPM_RC_ATTRIBUTES if the key is not a
83 // signing key but that was checked above. TPM_RC_VALUE would mean that the
84 // data to sign is too large but the data to sign is a digest
85 result = SignAttestInfo(in->signHandle,
86                        &in->inScheme,
87                        &certifyInfo,
88                        &in->qualifyingData,
89                        &out->certifyInfo,
90                        &out->signature);
91 if(result != TPM_RC_SUCCESS)
92     return result;
93
94 // orderly state should be cleared because of the reporting of clock info
95 // if signing happens
96 if(in->signHandle != TPM_RH_NULL)
97     g_clearOrderly = TRUE;
98
99 return TPM_RC_SUCCESS;
100 }

```