INTERNATIONAL STANDARD

ISO/IEC 11889-3

Second edition
2015-12-15

Information technology — Trusted Platform Module Library —

Part 3: Commands

Technologies de l'information — Bibliothèque de module de plate-forme de confiance — Partie 3: Commandes

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

Reference number
ISO/IEC 11889-3:2015(E)

© ISO/IEC 2015

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2015, Published in Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

 © ISO/IEC 2015 – All rights reserved

CONTENTS

Foreword xxiv
Introduction xxv
1 Scope 1
2 Normative references 2
3 Terms and Definitions 2
4 Symbols and abbreviated terms 2
5 Notation 2
5.1 Introduction 2
5.2 Table Decorations 2
5.3 Handle and Parameter Demarcation 4
5.4 AuthorizationSize and ParameterSize 4
6 Command Processing 5
6.1 Introduction 5
6.2 Command Header Validation 5
6.3 Mode Checks 5
6.4 Handle Area Validation 6
6.5 Session Area Validation 7
6.6 Authorization Checks 8
6.7 Parameter Decryption 10
6.8 Parameter Unmarshaling 10
6.8.1 Introduction 10
6.8.2 Unmarshaling Errors 10
6.9 Command Post Processing 11
7 Response Values 13
7.1 Tag 13
7.2 Response Codes 13
8 Implementation Dependent 16
9 Detailed Actions Assumptions 17
9.1 Introduction 17
9.2 Pre-processing 17
9.3 Post Processing 17
10 Start-up 18
10.1 Introduction 18
10.2 _TPM_Init 18
10.2.1 General Description 18
10.2.2 Detailed Actions 19
10.3 TPM2_Startup 20
10.3.1 General Description 20
10.3.2 Command and Response 23
10.3.3 Detailed Actions 24
10.4 TPM2_Shutdown 27
10.4.1 General Description 27

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

10.4.2 Command and Response 28
10.4.3 Detailed Actions 29

11 Testing 31

11.1 Introduction 31
11.2 TPM2_SelfTest 32

11.2.1 General Description 32
11.2.2 Command and Response 33
11.2.3 Detailed Actions 34

11.3 TPM2_IncrementalSelfTest 35

11.3.1 General Description 35
11.3.2 Command and Response 36
11.3.3 Detailed Actions 37

11.4 TPM2_GetTestResult 38

11.4.1 General Description 38
11.4.2 Command and Response 39
11.4.3 Detailed Actions 40

12 Session Commands 41

12.1 TPM2_StartAuthSession 41

12.1.1 General Description 41
12.1.2 Command and Response 43
12.1.3 Detailed Actions 44

12.2 TPM2_PolicyRestart 46

12.2.1 General Description 46
12.2.2 Command and Response 47
12.2.3 Detailed Actions 48

13 Object Commands 49

13.1 TPM2_Create 49

13.1.1 General Description 49
13.1.2 Command and Response 52
13.1.3 Detailed Actions 53

13.2 TPM2_Load 55

13.2.1 General Description 55
13.2.2 Command and Response 56
13.2.3 Detailed Actions 57

13.3 TPM2_LoadExternal 59

13.3.1 General Description 59
13.3.2 Command and Response 61
13.3.3 Detailed Actions 62

13.4 TPM2_ReadPublic 64

13.4.1 General Description 64
13.4.2 Command and Response 65

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

13.4.3 Detailed Actions 66
13.5 TPM2_ActivateCredential 67
13.5.1 General Description 67
13.5.2 Command and Response 68
13.5.3 Detailed Actions 69
13.6 TPM2_MakeCredential 71
13.6.1 General Description 71
13.6.2 Command and Response 72
13.6.3 Detailed Actions 73
13.7 TPM2_Unseal 74
13.7.1 General Description 74
13.7.2 Command and Response 75
13.7.3 Detailed Actions 76
13.8 TPM2_ObjectChangeAuth 77
13.8.1 General Description 77
13.8.2 Command and Response 78
13.8.3 Detailed Actions 79
14 Duplication Commands 81
14.1 TPM2_Duplicate 81
14.1.1 General Description 81
14.1.2 Command and Response 82
14.1.3 Detailed Actions 83
14.2 TPM2_Rewrap 85
14.2.1 General Description 85
14.2.2 Command and Response 86
14.2.3 Detailed Actions 87
14.3 TPM2_Import 90
14.3.1 General Description 90
14.3.2 Command and Response 92
14.3.3 Detailed Actions 93
15 Asymmetric Primitives 97
15.1 Introduction 97
15.2 TPM2_RSA_Encrypt 97
15.2.1 General Description 97
15.2.2 Command and Response 99
15.2.3 Detailed Actions 100
15.3 TPM2_RSA_Decrypt 102
15.3.1 General Description 102
15.3.2 Command and Response 103
15.3.3 Detailed Actions 104
15.4 TPM2_ECDH_KeyGen 106

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

15.4.1 General Description 106
15.4.2 Command and Response 107
15.4.3 Detailed Actions 108
15.5 TPM2_ECDH_ZGen 110
15.5.1 General Description 110
15.5.2 Command and Response 111
15.5.3 Detailed Actions 112
15.6 TPM2_ECC_Parameters 113
15.6.1 General Description 113
15.6.2 Command and Response 113
15.6.3 Detailed Actions 114
15.7 TPM2_ZGen_2Phase 114
15.7.1 General Description 114
15.7.2 Command and Response 116
15.7.3 Detailed Actions 117
16 Symmetric Primitives 119
16.1 Introduction 119
16.2 TPM2_EncryptDecrypt 121
16.2.1 General Description 121
16.2.2 Command and Response 122
16.2.3 Detailed Actions 123
16.3 TPM2_Hash 125
16.3.1 General Description 125
16.3.2 Command and Response 126
16.3.3 Detailed Actions 127
16.4 TPM2_HMAC 128
16.4.1 General Description 128
16.4.2 Command and Response 129
16.4.3 Detailed Actions 130
17 Random Number Generator 132
17.1 TPM2_GetRandom 132
17.1.1 General Description 132
17.1.2 Command and Response 133
17.1.3 Detailed Actions 134
17.2 TPM2_StirRandom 135
17.2.1 General Description 135
17.2.2 Command and Response 136
17.2.3 Detailed Actions 137
18 Hash/HMAC/Event Sequences 138
18.1 Introduction 138
18.2 TPM2_HMAC_Start 138

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

18.2.1 General Description 138
18.2.2 Command and Response 140
18.2.3 Detailed Actions 141
18.3 TPM2_HashSequenceStart 143
18.3.1 General Description 143
18.3.2 Command and Response 144
18.3.3 Detailed Actions 145
18.4 TPM2_SequenceUpdate 146
18.4.1 General Description 146
18.4.2 Command and Response 147
18.4.3 Detailed Actions 148
18.5 TPM2_SequenceComplete 150
18.5.1 General Description 150
18.5.2 Command and Response 151
18.5.3 Detailed Actions 152
18.6 TPM2_EventSequenceComplete 154
18.6.1 General Description 154
18.6.2 Command and Response 155
18.6.3 Detailed Actions 156
19 Attestation Commands 158
19.1 Introduction 158
19.2 TPM2_Certify 160
19.2.1 General Description 160
19.2.2 Command and Response 161
19.2.3 Detailed Actions 162
19.3 TPM2_CertifyCreation 164
19.3.1 General Description 164
19.3.2 Command and Response 165
19.3.3 Detailed Actions 166
19.4 TPM2_Quote 168
19.4.1 General Description 168
19.4.2 Command and Response 169
19.4.3 Detailed Actions 170
19.5 TPM2_GetSessionAuditDigest 172
19.5.1 General Description 172
19.5.2 Command and Response 173
19.5.3 Detailed Actions 174
19.6 TPM2_GetCommandAuditDigest 176
19.6.1 General Description 176
19.6.2 Command and Response 177
19.6.3 Detailed Actions 178
19.7 TPM2_GetTime 180

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

19.7.1 General Description 180
19.7.2 Command and Response 181
19.7.3 Detailed Actions 182

20 Ephemeral EC Keys 184

20.1 Introduction 184
20.2 TPM2_Commit 185

20.2.1 General Description 185
20.2.2 Command and Response 186
20.2.3 Detailed Actions 187

20.3 TPM2_EC_Ephemeral 190

20.3.1 General Description 190
20.3.2 Command and Response 191
20.3.3 Detailed Actions 192

21 Signing and Signature Verification 193

21.1 TPM2_VerifySignature 193

21.1.1 General Description 193
21.1.2 Command and Response 194
21.1.3 Detailed Actions 195

21.2 TPM2_Sign 197

21.2.1 General Description 197
21.2.2 Command and Response 198
21.2.3 Detailed Actions 199

22 Command Audit 201

22.1 Introduction 201
22.2 TPM2_SetCommandCodeAuditStatus 202

22.2.1 General Description 202
22.2.2 Command and Response 203
22.2.3 Detailed Actions 204

23 Integrity Collection (PCR) 206

23.1 Introduction 206
23.2 TPM2_PCR_Extend 207

23.2.1 General Description 207
23.2.2 Command and Response 208
23.2.3 Detailed Actions 209

23.3 TPM2_PCR_Event 210

23.3.1 General Description 210
23.3.2 Command and Response 211
23.3.3 Detailed Actions 212

23.4 TPM2_PCR_Read 214

23.4.1 General Description 214
23.4.2 Command and Response 215
23.4.3 Detailed Actions 216

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.5 TPM2_PCR_Allocate 217
23.5.1 General Description 217
23.5.2 Command and Response 218
23.5.3 Detailed Actions 219
23.6 TPM2_PCR_SetAuthPolicy 220
23.6.1 General Description 220
23.6.2 Command and Response 221
23.6.3 Detailed Actions 222
23.7 TPM2_PCR_SetAuthValue 223
23.7.1 General Description 223
23.7.2 Command and Response 224
23.7.3 Detailed Actions 225
23.8 TPM2_PCR_Reset 226
23.8.1 General Description 226
23.8.2 Command and Response 227
23.8.3 Detailed Actions 228
23.9 _TPM_Hash_Start 229
23.9.1 Description 229
23.9.2 Detailed Actions 230
23.10 _TPM_Hash_Data 231
23.10.1 Description 231
23.10.2 Detailed Actions 232
23.11 _TPM_Hash_End 233
23.11.1 Description 233
23.11.2 Detailed Actions 234
24 Enhanced Authorization (EA) Commands 236
24.1 Introduction 236
24.2 Signed Authorization Actions 237
24.2.1 Introduction 237
24.2.2 Policy Parameter Checks 237
24.2.3 Policy Digest Update Function (PolicyUpdate()) 238
24.2.4 Policy Context Updates 239
24.2.5 Policy Ticket Creation 240
24.3 TPM2_PolicySigned 241
24.3.1 General Description 241
24.3.2 Command and Response 243
24.3.3 Detailed Actions 244
24.4 TPM2_PolicySecret 247
24.4.1 General Description 247
24.4.2 Command and Response 248
24.4.3 Detailed Actions 249

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.5 TPM2_PolicyTicket 251
24.5.1 General Description 251
24.5.2 Command and Response 252
24.5.3 Detailed Actions 253
24.6 TPM2_PolicyOR 255
24.6.1 General Description 255
24.6.2 Command and Response 256
24.6.3 Detailed Actions 257
24.7 TPM2_PolicyPCR 259
24.7.1 General Description 259
24.7.2 Command and Response 261
24.7.3 Detailed Actions 262
24.8 TPM2_PolicyLocality 264
24.8.1 General Description 264
24.8.2 Command and Response 265
24.8.3 Detailed Actions 266
24.9 TPM2_PolicyNV 268
24.9.1 General Description 268
24.9.2 Command and Response 269
24.9.3 Detailed Actions 270
24.10 TPM2_PolicyCounterTimer 273
24.10.1 General Description 273
24.10.2 Command and Response 274
24.10.3 Detailed Actions 275
24.11 TPM2_PolicyCommandCode 278
24.11.1 General Description 278
24.11.2 Command and Response 279
24.11.3 Detailed Actions 280
24.12 TPM2_PolicyPhysicalPresence 281
24.12.1 General Description 281
24.12.2 Command and Response 282
24.12.3 Detailed Actions 283
24.13 TPM2_PolicyCpHash 284
24.13.1 General Description 284
24.13.2 Command and Response 285
24.13.3 Detailed Actions 286
24.14 TPM2_PolicyNameHash 288
24.14.1 General Description 288
24.14.2 Command and Response 289
24.14.3 Detailed Actions 290
24.15 TPM2_PolicyDuplicationSelect 292

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

24.15.1 General Description 292
24.15.2 Command and Response 293
24.15.3 Detailed Actions 294
24.16 TPM2_PolicyAuthorize 296
24.16.1 General Description 296
24.16.2 Command and Response 297
24.16.3 Detailed Actions 298
24.17 TPM2_PolicyAuthValue 300
24.17.1 General Description 300
24.17.2 Command and Response 301
24.17.3 Detailed Actions 302
24.18 TPM2_PolicyPassword 303
24.18.1 General Description 303
24.18.2 Command and Response 304
24.18.3 Detailed Actions 305
24.19 TPM2_PolicyGetDigest 306
24.19.1 General Description 306
24.19.2 Command and Response 307
24.19.3 Detailed Actions 308
24.20 TPM2_PolicyNvWritten 309
24.20.1 General Description 309
24.20.2 Command and Response 310
24.20.3 Detailed Actions 311
25 Hierarchy Commands 313
25.1 TPM2_CreatePrimary 313
25.1.1 General Description 313
25.1.2 Command and Response 314
25.1.3 Detailed Actions 315
25.2 TPM2_HierarchyControl 317
25.2.1 General Description 317
25.2.2 Command and Response 318
25.2.3 Detailed Actions 319
25.3 TPM2_SetPrimaryPolicy 321
25.3.1 General Description 321
25.3.2 Command and Response 322
25.3.3 Detailed Actions 323
25.4 TPM2_ChangePPS 325
25.4.1 General Description 325
25.4.2 Command and Response 326
25.4.3 Detailed Actions 327
25.5 TPM2_ChangeEPS 328

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.5.1 General Description 328
25.5.2 Command and Response 329
25.5.3 Detailed Actions 330
25.6 TPM2_Clear 331
25.6.1 General Description 331
25.6.2 Command and Response 332
25.6.3 Detailed Actions 333
25.7 TPM2_ClearControl 335
25.7.1 General Description 335
25.7.2 Command and Response 336
25.7.3 Detailed Actions 337
25.8 TPM2_HierarchyChangeAuth 338
25.8.1 General Description 338
25.8.2 Command and Response 339
25.8.3 Detailed Actions 340
26 Dictionary Attack Functions 341
26.1 Introduction 341
26.2 TPM2_DictionaryAttackLockReset 341
26.2.1 General Description 341
26.2.2 Command and Response 342
26.2.3 Detailed Actions 343
26.3 TPM2_DictionaryAttackParameters 344
26.3.1 General Description 344
26.3.2 Command and Response 345
26.3.3 Detailed Actions 346
27 Miscellaneous Management Functions 347
27.1 Introduction 347
27.2 TPM2_PP_Commands 347
27.2.1 General Description 347
27.2.2 Command and Response 348
27.2.3 Detailed Actions 349
27.3 TPM2_SetAlgorithmSet 350
27.3.1 General Description 350
27.3.2 Command and Response 351
27.3.3 Detailed Actions 352
28 Field Upgrade 353
28.1 Introduction 353
28.2 TPM2_FieldUpgradeStart 355
28.2.1 General Description 355
28.2.2 Command and Response 356
28.2.3 Detailed Actions 357
28.3 TPM2_FieldUpgradeData 358

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

28.3.1 General Description 358
28.3.2 Command and Response 359
28.3.3 Detailed Actions 360
28.4 TPM2_FirmwareRead 361
28.4.1 General Description 361
28.4.2 Command and Response 362
28.4.3 Detailed Actions 363
29 Context Management 364
29.1 Introduction 364
29.2 TPM2_ContextSave 364
29.2.1 General Description 364
29.2.2 Command and Response 365
29.2.3 Detailed Actions 366
29.3 TPM2_ContextLoad 369
29.3.1 General Description 369
29.3.2 Command and Response 370
29.3.3 Detailed Actions 371
29.4 TPM2_FlushContext 374
29.4.1 General Description 374
29.4.2 Command and Response 375
29.4.3 Detailed Actions 376
29.5 TPM2_EvictControl 377
29.5.1 General Description 377
29.5.2 Command and Response 379
29.5.3 Detailed Actions 380
30 Clocks and Timers 382
30.1 TPM2_ReadClock 382
30.1.1 General Description 382
30.1.2 Command and Response 383
30.1.3 Detailed Actions 384
30.2 TPM2_ClockSet 385
30.2.1 General Description 385
30.2.2 Command and Response 386
30.2.3 Detailed Actions 387
30.3 TPM2_ClockRateAdjust 388
30.3.1 General Description 388
30.3.2 Command and Response 389
30.3.3 Detailed Actions 390
31 Capability Commands 391
31.1 Introduction 391
31.2 TPM2_GetCapability 391

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

31.2.1 General Description 391
31.2.2 Command and Response 395
31.2.3 Detailed Actions 396
31.3 TPM2_TestParms 399
31.3.1 General Description 399
31.3.2 Command and Response 400
31.3.3 Detailed Actions 401
32 Non-volatile Storage 402
32.1 Introduction 402
32.2 NV Counters 404
32.3 TPM2_NV_DefineSpace 405
32.3.1 General Description 405
32.3.2 Command and Response 407
32.3.3 Detailed Actions 408
32.4 TPM2_NV_UndefineSpace 411
32.4.1 General Description 411
32.4.2 Command and Response 412
32.4.3 Detailed Actions 413
32.5 TPM2_NV_UndefineSpaceSpecial 414
32.5.1 General Description 414
32.5.2 Command and Response 415
32.5.3 Detailed Actions 416
32.6 TPM2_NV_ReadPublic 417
32.6.1 General Description 417
32.6.2 Command and Response 418
32.6.3 Detailed Actions 419
32.7 TPM2_NV_Write 420
32.7.1 General Description 420
32.7.2 Command and Response 421
32.7.3 Detailed Actions 422
32.8 TPM2_NV_Increment 424
32.8.1 General Description 424
32.8.2 Command and Response 425
32.8.3 Detailed Actions 426
32.9 TPM2_NV_Extend 428
32.9.1 General Description 428
32.9.2 Command and Response 429
32.9.3 Detailed Actions 430
32.10 TPM2_NV_SetBits 432
32.10.1 General Description 432
32.10.2 Command and Response 433
32.10.3 Detailed Actions 434

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

32.11 TPM2_NV_WriteLock 436
32.11.1 General Description 436
32.11.2 Command and Response 437
32.11.3 Detailed Actions 438
32.12 TPM2_NV_GlobalWriteLock 440
32.12.1 General Description 440
32.12.2 Command and Response 441
32.12.3 Detailed Actions 442
32.13 TPM2_NV_Read 443
32.13.1 General Description 443
32.13.2 Command and Response 444
32.13.3 Detailed Actions 445
32.14 TPM2_NV_ReadLock 446
32.14.1 General Description 446
32.14.2 Command and Response 447
32.14.3 Detailed Actions 448
32.15 TPM2_NV_ChangeAuth 450
32.15.1 General Description 450
32.15.2 Command and Response 451
32.15.3 Detailed Actions 452
32.16 TPM2_NV_Certify 453
32.16.1 General Description 453
32.16.2 Command and Response 454
32.16.3 Detailed Actions 455
Bibliography 457

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

Tables

Table 1 — Command Modifiers and Decoration 3
Table 2 — Separators 4
Table 3 — Unmarshaling Errors 11
Table 4 — Command-Independent Response Codes 14
Table 5 — TPM2_Startup Command 23
Table 6 — TPM2_Startup Response 23
Table 7 — TPM2_Startup Errors 24
Table 8 — TPM2_Shutdown Command 28
Table 9 — TPM2_Shutdown Response 28
Table 10 — TPM2_Shutdown Errors 29
Table 11 — TPM2_SelfTest Command 33
Table 12 — TPM2_SelfTest Response 33
Table 13 — TPM2_SelfTest Errors 34
Table 14 — TPM2_IncrementalSelfTest Command 36
Table 15 — TPM2_IncrementalSelfTest Response 36
Table 16 — TPM2_IncrementalSelfTest Errors 37
Table 17 — TPM2_GetTestResult Command 39
Table 18 — TPM2_GetTestResult Response 39
Table 19 — TPM2_StartAuthSession Command 43
Table 20 — TPM2_StartAuthSession Response 43
Table 21 — TPM2_StartAuthSession Errors 44
Table 22 — TPM2_PolicyRestart Command 47
Table 23 — TPM2_PolicyRestart Response 47
Table 24 — TPM2_Create Command 52
Table 25 — TPM2_Create Response 52
Table 26 — TPM2_Create Errors 53
Table 27 — TPM2_Load Command 56
Table 28 — TPM2_Load Response 56
Table 29 — TPM2_Load Errors 57
Table 30 — TPM2_LoadExternal Command 61
Table 31 — TPM2_LoadExternal Response 61
Table 32 — TPM2_LoadExternal Errors 62
Table 33 — TPM2_ReadPublic Command 65
Table 34 — TPM2_ReadPublic Response 65
Table 35 — TPM2_ReadPublic Errors 66
Table 36 — TPM2_ActivateCredential Command 68
Table 37 — TPM2_ActivateCredential Response 68
Table 38 — TPM2_ActivateCredential Errors 69

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

Table 39 — TPM2_MakeCredential Command 72
Table 40 — TPM2_MakeCredential Response 72
Table 41 — TPM2_MakeCredential Errors 73
Table 42 — TPM2_Unseal Command 75
Table 43 — TPM2_Unseal Response 75
Table 44 — TPM2_Unseal Errors 76
Table 45 — TPM2_ObjectChangeAuth Command 78
Table 46 — TPM2_ObjectChangeAuth Response 78
Table 47 — TPM2_ObjectChangeAuth Errors 79
Table 48 — TPM2_Duplicate Command 82
Table 49 — TPM2_Duplicate Response 82
Table 50 — TPM2_Duplicate Errors 83
Table 51 — TPM2_Rewrap Command 86
Table 52 — TPM2_Rewrap Response 86
Table 53 — TPM2_Rewrap Errors 87
Table 54 — TPM2_Import Command 92
Table 55 — TPM2_Import Response 92
Table 56 — TPM2_Import Errors 93
Table 57 — Padding Scheme Selection 97
Table 58 — Message Size Limits Based on Padding 98
Table 59 — TPM2_RSA_Encrypt Command 99
Table 60 — TPM2_RSA_Encrypt Response 99
Table 61 — TPM2_RSA_Encrypt Errors 100
Table 62 — TPM2_RSA_Decrypt Command 103
Table 63 — TPM2_RSA_Decrypt Response 103
Table 64 — TPM2_RSA_Decrypt Errors 104
Table 65 — TPM2_ECDH_KeyGen Command 107
Table 66 — TPM2_ECDH_KeyGen Response 107
Table 67 — TPM2_ECDH_KeyGen Errors 108
Table 68 — TPM2_ECDH_ZGen Command 111
Table 69 — TPM2_ECDH_ZGen Response 111
Table 70 — TPM2_ECDH_ZGen Errors 112
Table 71 — TPM2_ECC_Parameters Command 113
Table 72 — TPM2_ECC_Parameters Response 113
Table 73 — TPM2_ECC_Parameters Errors 114
Table 74 — TPM2_ZGen_2Phase Command 116
Table 75 — TPM2_ZGen_2Phase Response 116
Table 76 — TPM2_ZGen_2Phase Errors 117
Table 77 — Symmetric Chaining Process 120

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

Table 78 — TPM2_EncryptDecrypt Command 122
Table 79 — TPM2_EncryptDecrypt Response 122
Table 80 — TPM2_EncryptDecrypt Errors 123
Table 81 — TPM2_Hash Command 126
Table 82 — TPM2_Hash Response 126
Table 83 — TPM2_HMAC Command 129
Table 84 — TPM2_HMAC Response 129
Table 85 — TPM2_HMAC Errors 130
Table 86 — TPM2_GetRandom Command 133
Table 87 — TPM2_GetRandom Response 133
Table 88 — TPM2_StirRandom Command 136
Table 89 — TPM2_StirRandom Response 136
Table 90 — Hash Selection Matrix 138
Table 91 — TPM2_HMAC_Start Command 140
Table 92 — TPM2_HMAC_Start Response 140
Table 93 — TPM2_HMAC_Start Errors 141
Table 94 — TPM2_HashSequenceStart Command 144
Table 95 — TPM2_HashSequenceStart Response 144
Table 96 — TPM2_HashSequenceStart Errors 145
Table 97 — TPM2_SequenceUpdate Command 147
Table 98 — TPM2_SequenceUpdate Response 147
Table 99 — TPM2_SequenceUpdate Errors 148
Table 100 — TPM2_SequenceComplete Command 151
Table 101 — TPM2_SequenceComplete Response 151
Table 102 — TPM2_SequenceComplete Errors 152
Table 103 — TPM2_EventSequenceComplete Command 155
Table 104 — TPM2_EventSequenceComplete Response 155
Table 105 — TPM2_EventSequenceComplete Errors 156
Table 106 — TPM2_Certify Command 161
Table 107 — TPM2_Certify Response 161
Table 108 — TPM2_Certify Errors 162
Table 109 — TPM2_CertifyCreation Command 165
Table 110 — TPM2_CertifyCreation Response 165
Table 111 — TPM2_CertifyCreation Errors 166
Table 112 — TPM2_Quote Command 169
Table 113 — TPM2_Quote Response 169
Table 114 — TPM2_Quote Errors 170
Table 115 — TPM2_GetSessionAuditDigest Command 173
Table 116 — TPM2_GetSessionAuditDigest Response 173

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

Table 117 — TPM2_GetSessionAuditDigest Errors 174
Table 118 — TPM2_GetCommandAuditDigest Command 177
Table 119 — TPM2_GetCommandAuditDigest Response 177
Table 120 — TPM2_GetCommandAuditDigest Errors 178
Table 121 — TPM2_GetTime Command 181
Table 122 — TPM2_GetTime Response 181
Table 123 — TPM2_GetTime Errors 182
Table 124 — TPM2_Commit Command 186
Table 125 — TPM2_Commit Response 186
Table 126 — TPM2_Commit Response Errors 187
Table 127 — TPM2_EC_Ephemeral Command 191
Table 128 — TPM2_EC_Ephemeral Response 191
Table 129 — TPM2_VerifySignature Command 194
Table 130 — TPM2_VerifySignature Response 194
Table 131 — TPM2_VerifySignature Errors 195
Table 132 — TPM2_Sign Command 198
Table 133 — TPM2_Sign Response 198
Table 134 — TPM2_Sign Response Errors 199
Table 135 — TPM2_SetCommandCodeAuditStatus Command 203
Table 136 — TPM2_SetCommandCodeAuditStatus Response 203
Table 137 — TPM2_PCR_Extend Command 208
Table 138 — TPM2_PCR_Extend Response 208
Table 139 — TPM2_PCR_Extend Errors 209
Table 140 — TPM2_PCR_Event Command 211
Table 141 — TPM2_PCR_Event Response 211
Table 142 — TPM2_PCR_Event Errors 212
Table 143 — TPM2_PCR_Read Command 215
Table 144 — TPM2_PCR_Read Response 215
Table 145 — TPM2_PCR_Allocate Command 218
Table 146 — TPM2_PCR_Allocate Response 218
Table 147 — TPM2_PCR_Allocate Errors 219
Table 148 — TPM2_PCR_SetAuthPolicy Command 221
Table 149 — TPM2_PCR_SetAuthPolicy Response 221
Table 150 — TPM2_PCR_SetAuthPolicy Errors 222
Table 151 — TPM2_PCR_SetAuthValue Command 224
Table 152 — TPM2_PCR_SetAuthValue Response 224
Table 153 — TPM2_PCR_SetAuthValue Errors 225
Table 154 — TPM2_PCR_Reset Command 227
Table 155 — TPM2_PCR_Reset Response 227

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

Table 156 — TPM2_PCR_Reset Errors 228
Table 157 — TPM2_PolicySigned Command 243
Table 158 — TPM2_PolicySigned Response 243
Table 159 — TPM2_PolicySigned Errors 244
Table 160 — TPM2_PolicySecret Command 248
Table 161 — TPM2_PolicySecret Response 248
Table 162 — TPM2_PolicySecret Errors 249
Table 163 — TPM2_PolicyTicket Command 252
Table 164 — TPM2_PolicyTicket Response 252
Table 165 — TPM2_PolicyTicket Errors 253
Table 166 — TPM2_PolicyOR Command 256
Table 167 — TPM2_PolicyOR Response 256
Table 168 — TPM2_PolicyOR Errors 257
Table 169 — TPM2_PolicyPCR Command 261
Table 170 — TPM2_PolicyPCR Response 261
Table 171 — TPM2_PolicyPCR Errors 262
Table 172 — TPM2_PolicyLocality Command 265
Table 173 — TPM2_PolicyLocality Response 265
Table 174 — TPM2_PolicyLocality Errors 266
Table 175 — TPM2_PolicyNV Command 269
Table 176 — TPM2_PolicyNV Response 269
Table 177 — TPM2_PolicyNV Errors 270
Table 178 — TPM2_PolicyCounterTimer Command 274
Table 179 — TPM2_PolicyCounterTimer Response 274
Table 180 — TPM2_PolicyCounterTimer Errors 275
Table 181 — TPM2_PolicyCommandCode Command 279
Table 182 — TPM2_PolicyCommandCode Response 279
Table 183 — TPM2_PolicyCommandCode Errors 280
Table 184 — TPM2_PolicyPhysicalPresence Command 282
Table 185 — TPM2_PolicyPhysicalPresence Response 282
Table 186 — TPM2_PolicyCpHash Command 285
Table 187 — TPM2_PolicyCpHash Response 285
Table 188 — TPM2_PolicyCpHash Errors 286
Table 189 — TPM2_PolicyNameHash Command 289
Table 190 — TPM2_PolicyNameHash Response 289
Table 191 — TPM2_PolicyNameHash Errors 290
Table 192 — TPM2_PolicyDuplicationSelect Command 293
Table 193 — TPM2_PolicyDuplicationSelect Response 293
Table 194 — TPM2_PolicyDuplicationSelect Errors 294

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

Table 195 — TPM2_PolicyAuthorize Command 297
Table 196 — TPM2_PolicyAuthorize Response 297
Table 197 — TPM2_PolicyAuthorize Errors 298
Table 198 — TPM2_PolicyAuthValue Command 301
Table 199 — TPM2_PolicyAuthValue Response 301
Table 200 — TPM2_PolicyPassword Command 304
Table 201 — TPM2_PolicyPassword Response 304
Table 202 — TPM2_PolicyGetDigest Command 307
Table 203 — TPM2_PolicyGetDigest Response 307
Table 204 — TPM2_PolicyNvWritten Command 310
Table 205 — TPM2_PolicyNvWritten Response 310
Table 206 — TPM2_PolicyNvWritten Errors 311
Table 207 — TPM2_CreatePrimary Command 314
Table 208 — TPM2_CreatePrimary Response 314
Table 209 — TPM2_CreatePrimary Errors 315
Table 210 — TPM2_HierarchyControl Command 318
Table 211 — TPM2_HierarchyControl Response 318
Table 212 — TPM2_HierarchyControl Errors 319
Table 213 — TPM2_SetPrimaryPolicy Command 322
Table 214 — TPM2_SetPrimaryPolicy Response 322
Table 215 — TPM2_SetPrimaryPolicy Errors 323
Table 216 — TPM2_ChangePPS Command 326
Table 217 — TPM2_ChangePPS Response 326
Table 218 — TPM2_ChangeEPS Command 329
Table 219 — TPM2_ChangeEPS Response 329
Table 220 — TPM2_Clear Command 332
Table 221 — TPM2_Clear Response 332
Table 222 — TPM2_Clear Errors 333
Table 223 — TPM2_ClearControl Command 336
Table 224 — TPM2_ClearControl Response 336
Table 225 — TPM2_ClearControl Errors 337
Table 226 — TPM2_HierarchyChangeAuth Command 339
Table 227 — TPM2_HierarchyChangeAuth Response 339
Table 228 — TPM2_HierarchyChangeAuth Errors 340
Table 229 — TPM2_DictionaryAttackLockReset Command 342
Table 230 — TPM2_DictionaryAttackLockReset Response 342
Table 231 — TPM2_DictionaryAttackParameters Command 345
Table 232 — TPM2_DictionaryAttackParameters Response 345
Table 233 — TPM2_PP_Commands Command 348

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

Table 234 — TPM2_PP_Commands Response 348
Table 235 — TPM2_SetAlgorithmSet Command 351
Table 236 — TPM2_SetAlgorithmSet Response 351
Table 237 — TPM2_FieldUpgradeStart Command 356
Table 238 — TPM2_FieldUpgradeStart Response 356
Table 239 — TPM2_FieldUpgradeData Command 359
Table 240 — TPM2_FieldUpgradeData Response 359
Table 241 — TPM2_FirmwareRead Command 362
Table 242 — TPM2_FirmwareRead Response 362
Table 243 — TPM2_ContextSave Command 365
Table 244 — TPM2_ContextSave Response 365
Table 245 — TPM2_ContextSave Errors 366
Table 246 — TPM2_ContextLoad Command 370
Table 247 — TPM2_ContextLoad Response 370
Table 248 — TPM2_ContextLoad Errors 371
Table 249 — TPM2_FlushContext Command 375
Table 250 — TPM2_FlushContext Response 375
Table 251 — TPM2_FlushContext Errors 376
Table 252 — TPM2_EvictControl Command 379
Table 253 — TPM2_EvictControl Response 379
Table 254 — TPM2_EvictControl Errors 380
Table 255 — TPM2_ReadClock Command 383
Table 256 — TPM2_ReadClock Response 383
Table 257 — TPM2_ClockSet Command 386
Table 258 — TPM2_ClockSet Response 386
Table 259 — TPM2_ClockSet Errors 387
Table 260 — TPM2_ClockRateAdjust Command 389
Table 261 — TPM2_ClockRateAdjust Response 389
Table 262 — TPM2_GetCapability Command 395
Table 263 — TPM2_GetCapability Response 395
Table 264 — TPM2_GetCapability Errors 396
Table 265 — TPM2_TestParms Command 400
Table 266 — TPM2_TestParms Response 400
Table 267 — TPM2_NV_DefineSpace Command 407
Table 268 — TPM2_NV_DefineSpace Response 407
Table 269 — TPM2_NV_DefineSpace Errors 408
Table 270 — TPM2_NV_UndefineSpace Command 412
Table 271 — TPM2_NV_UndefineSpace Response 412
Table 272 — TPM2_NV_UndefineSpace Errors 413

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

Table 273 — TPM2_NV_UndefineSpaceSpecial Command 415
Table 274 — TPM2_NV_UndefineSpaceSpecial Response 415
Table 275 — TPM2_NV_UndefineSpaceSpecial Errors 416
Table 276 — TPM2_NV_ReadPublic Command 418
Table 277 — TPM2_NV_ReadPublic Response 418
Table 278 — TPM2_NV_Write Command 421
Table 279 — TPM2_NV_Write Response 421
Table 280 — TPM2_NV_Write Errors 422
Table 281 — TPM2_NV_Increment Command 425
Table 282 — TPM2_NV_Increment Response 425
Table 283 — TPM2_NV_Increment Errors 426
Table 284 — TPM2_NV_Extend Command 429
Table 285 — TPM2_NV_Extend Response 429
Table 286 — TPM2_NV_Extend Errors 430
Table 287 — TPM2_NV_SetBits Command 433
Table 288 — TPM2_NV_SetBits Response 433
Table 289 — TPM2_NV_SetBits Errors 434
Table 290 — TPM2_NV_WriteLock Command 437
Table 291 — TPM2_NV_WriteLock Response 437
Table 292 — TPM2_NV_WriteLock Errors 438
Table 293 — TPM2_NV_GlobalWriteLock Command 441
Table 294 — TPM2_NV_GlobalWriteLock Response 441
Table 295 — TPM2_NV_Read Command 444
Table 296 — TPM2_NV_Read Response 444
Table 297 — TPM2_NV_Read Errors 445
Table 298 — TPM2_NV_ReadLock Command 447
Table 299 — TPM2_NV_ReadLock Response 447
Table 300 — TPM2_NV_ReadLock Errors 448
Table 301 — TPM2_NV_ChangeAuth Command 451
Table 302 — TPM2_NV_ChangeAuth Response 451
Table 303 — TPM2_NV_ChangeAuth Errors 452
Table 304 — TPM2_NV_Certify Command 454
Table 305 — TPM2_NV_Certify Response 454
Table 306 — TPM2_NV_Certify Errors 455

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the WTO principles in the Technical Barriers to Trade (TBT), see the following URL: Foreword — Supplementary information.

ISO/IEC 11889-3 was prepared by the Trusted Computing Group (TCG) and was adopted, under the PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

This second edition cancels and replaces the first edition (ISO/IEC 11889-3:2009), which has been technically revised.

ISO/IEC 11889 consists of the following parts, under the general title *Information technology — Trusted Platform Module Library*:

– *Part 1: Architecture*

– *Part 2: Structures*

– *Part 3: Commands*

– *Part 4: Supporting routines*

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

Introduction

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of a patent.

ISO and IEC take no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured the ISO and IEC that he/she is willing to negotiate licences either free of charge or under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO and IEC. Information may be obtained from:

Fujitsu Limited **1-1, Kamikodanaka 4-chrome, Nakahara-ku, Kawasaki-shi, Kanagawa, 211-8588 Japan**
Microsoft Corporation **One Microsoft Way, Redmond, WA 98052**
Enterasys Networks, Inc **50 Minuteman Road, US-Andover, MA 01810**
Lenovo **1009 Think Place, US-Morrisville, NC 27560-8496**
Advanced Micro devices, Inc. - AMD **7171 Southwest Parkway, Mailstop B100.3, US-Austin, Texas 78735**
Hewlett-Packard Company **P.O. Box 10490, US-Palo Alto, CA 94303-0969**
Infineon Technologies AG - Neubiberg **Am Campeon 1-12, DE-85579 Neubiberg**
Sun Microsystems Inc. - Menlo Park, CA **10 Network Circle, UMPK10-146, US-Menlo Park, CA 94025**
IBM Corporation **North Castle Drive, US-Armonk, N.Y. 10504**
Intel Corporation **5200 Elam Young Parkway, US-Hillsboro, OR 97123**

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified above. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO (www.iso.org/patents) and IEC (http://patents.iec.ch) maintain on-line databases of patents relevant to their standards. Users are encouraged to consult the databases for the most up to date information concerning patents.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

Information technology — Trusted Platform Module Library — Part 3: Commands

1 Scope

This part of ISO/IEC 11889 contains the definitions of the Trusted Platform Module (TPM) commands. These commands make use of the constants, flags, structures, and union definitions defined in ISO/IEC 11889-2.

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 of ISO/IEC 11889 is normative but does not fully describe the behavior of a TPM. The combination of this part of ISO/IEC 11889 and ISO/IEC 11889-4 is sufficient to fully describe the required behavior of a TPM.

The code this part of ISO/IEC 11889 and ISO/IEC 11889-4 is written to define the behavior of a compliant TPM. In some cases 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 this part of ISO/IEC 11889 would be compliant.

EXAMPLE Firmware update is a case where it is not possible to provide a compliant implementation.

The code in thie part of ISO/IEC 11889 and ISO/IEC 11889-4 is not written to meet any particular level of conformance nor does this specification require that a TPM meet any particular level of conformance.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

- ISO/IEC 11889-1, *Information technology — Trusted Platform Module Library — Part 1: Architecture*
- ISO/IEC 11889-2, *Information technology — Trusted Platform Module Library — Part 2: Structures*
- ISO/IEC 11889-4, *Information technology — Trusted Platform Module Library — Part 4: Supporting routines*
- TCG Vendor ID Registry, available at <http://www.trustedcomputinggroup.org/resources/vendor_id_registry>

3 Terms and Definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 11889-1 apply.

4 Symbols and abbreviated terms

For the purposes of this document, the symbols and abbreviated terms given in ISO/IEC 11889-1 apply.

5 Notation

5.1 Introduction

For the purposes of this document, the notation given in ISO/IEC 11889-1 applies.

Command and response tables use various decorations to indicate the fields of the command and the allowed types. These decorations are specified in clause 5.

5.2 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

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 ISO/IEC 11889-2, clause 5.8, "Conditional Types"). 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 *platformAuth*/*platformPolicy* is provided.
+{PP}	A Description modifier – This modifier may follow TPM_RH_PLATFORM to indicate that Physical Presence may be required when *platformAuth*/*platformPolicy* is provided. The commands with this notation may be in the *setList* or *clearList* of TPM2_PP_Commands().
{NV}	A Description modifier – This modifier may follow the *commandCode* 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 a write to NV memory does not occur as part of the command actions.
{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 *commandCode* 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.
{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 *commandCode* 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.
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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

Notation	Meaning
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 *userWithAuth* in the Object's attributes. If the handle is TPM_RH_OWNER, TPM_RH_ENDORSEMENT, or TPM_RH_PLATFORM, userWithAuth is SET. If the Auth Role is ADMIN and the handle is an Object, the type of authorization is determined by the setting of *adminWithPolicy* in the Object's attributes. If the handle is TPM_RH_OWNER, TPM_RH_ENDORSEMENT, or TPM_RH_PLATFORM, adminWithPolicy is SET. 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. If the handle references an NV Index, then the allowed authorizations are determined by the settings of the attributes of the NV Index as specified in ISO/IEC 11889-2, clause 14.2, "TPMA_NV (NV Index Attributes)."

NOTE 1 Regarding {NV}, any command that uses authorization can cause a write to NV if there is an authorization failure. A TPM might use the occasion of command execution to update the NV copy of clock.

EXAMPLE 1 An example of combining the {NV} and {F} modifiers is {NV F}.

EXAMPLE 2 Regarding {F}, TPM2_SequenceComplete() will flush the context associated with the *sequenceHandle*.

EXAMPLE 3 An example of combining the {NV} and {E} modifiers is {NV E}.

EXAMPLE 4 Regarding {E}, TPM2_Clear() will flush all contexts associated with the Storage hierarchy and the Endorsement hierarchy.

EXAMPLE 5 Regarding Auth Role, TPM2_Certify needs the ADMIN role for the first handle (*objectHandle*). The policy authorization for objectHandle needs to contain TPM2_PolicyCommandCode(*commandCode* == TPM_CC_Certify). This sets the state of the policy so that it can be used for ADMIN role authorization in TPM2_Certify().

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

5.3 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

5.4 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.

 © ISO/IEC 2015 – All rights reserved

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 authorization is required to use the TPM entity associated with a handle, then at least one session will be present. To indicate this, the command *tag* Description field contains TPM_ST_SESSIONS. Addional sessions for audit, encrypt, and decrypt may be present..

When the command *tag* Description field contains TPM_ST_NO_SESSIONS, then no sessions are allowed and the *authorizationSize* field is not present.

When a command allows use of sessions when not required, the command *tag* Description field will indicate the types of sessions that may be used with the command.

6 Command Processing

6.1 Introduction

Clause 6 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.

6.2 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 for the command reported by the hardware process shall exactly match the value in *commandSize* (TPM_RC_COMMAND_SIZE).

NOTE A TPM can 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).

6.3 Mode Checks

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

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 possible.

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 can enter Failure mode during _TPM_Init processing, before TPM2_Startup(). Since the platform firmware cannot know that the TPM is in Failure mode without accessing it, and since the first command needs 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.

There can be failures where a TPM cannot record that it received TPM2_Startup(). In those cases, a TPM in failure mode can process TPM2_GetTestResult(), TPM2_GetCapability(), or the field upgrade commands. As a side effect, that TPM might process TPM2_GetTestResult(), TPM2_GetCapability() or the field upgrade commands before TPM2_Startup().

This is a corner case exception to the rule that TPM2_Startup() must be the first command.

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

6.4 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.

NOTE 1 A TPM needs to perform the handle area validation before the authorization checks because an authorization cannot be performed unless the authorization values and attributes for the referenced entity are known by the TPM. For them to be known, the referenced entity needs to be in the TPM and accessible.

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 2 The TPM can unmarshal a handle and validate that it references an entity on the TPM before unmarshaling a subsequent handle.

NOTE 3 If the submitted command contains fewer handles than required by the syntax of the command, the TPM can 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 handle in the command).

NOTE 4 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 hierarchy associated with the object (platform or storage, based on the handle value) is enabled (TPM_RC_HANDLE);

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

ii) the handle shall reference a persistent object that is currently in TPM non-volatile memory (TPM_RC_HANDLE);

iii) if the handle references a persistent object that is associated with the endorsement hierarchy, that the endorsement hierarchy is not disabled (TPM_RC_HANDLE); and

NOTE 5 The reference implementation keeps an internal attribute, passed down from a primary key to its descendents, indicating the object's hierarchy.

iv) 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 existing NV Index is not disabled (TPM_RC_HANDLE).

iii) If the command requires write access to the index data then TPMA_NV_WRITELOCKED is not SET (TPM_RC_LOCKED)

iv) If the command requires read access to the index data then TPMA_NV_READLOCKED is not SET (TPM_RC_LOCKED)

4) If the handle references a session, then the session context shall be present in TPM memory (TPM_RC_REFERENCE_S0 + N).

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 6 In the reference implementation, this TPM_RC_VALUE is returned by the unmarshaling code for a TPMI_DH_PCR.

6.5 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 commandSize – (sizeof(TPM_ST) + sizeof(UINT32) + sizeof(TPM_CC) + (N * sizeof(TPM_HANDLE)) + 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. The first session is session zero, N = 0.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

NOTE 2 If the HMAC and policy session contexts use the same memory, the type of the context needs to 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) 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 be 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, ora 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 can also be used for encrypting a response parameter.

5) An authorization session is present for each of the handles with the "@" decoration (TPM_RC_AUTH_MISSING).

6.6 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 needed 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 possible.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

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 need 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);

6) if *policySession->commandLocality* has been set, it shall match the locality of the command (TPM_RC_LOCALITY), and

7) 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 A policy session might 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 state other than *lockoutCount*.

NOTE 6 The TPM can decrease failedTries regardless of any other processing performed by the TPM. That is, the TPM can exit Lockout mode, regardless of the return code.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

6.7 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.

6.8 Parameter Unmarshaling

6.8.1 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 does not need to process parameters in this manner or to separate the parameter parsing from the command actions. This method was chosen for this part of ISO/IEC 11889 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 ISO/IEC 11889-2.

6.8.2 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. This is optional.

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

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 ISO/IEC 11889-2 definition of the parameter type even if that parameter is not used in the command actions.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

6.9 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.

If authorization HMAC computations are performed on the response, the HMAC keys used in the response will be the same as the HMAC keys used in processing the HMAC in the command.

© ISO/IEC 2015 – All rights reserved

NOTE 1 This primarily affects authorizations associated with a first write to an NV Index using a bound session. The computation of the HMAC in the response is performed as if the Name of the Index did not change as a consequence of the command actions. The session binding to the NV Index will not persist to any subsequent command.

NOTE 2 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 3 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

7 Response Values

7.1 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 an implementation of ISO/IEC 11889 (first edition) but the TPM is not capable of executing ISO/IEC 11889 (first edition) commands. So that the ISO/IEC 11889 (first edition) compatible software will have a recognizable response, the TPM sets *tag* to TPM_ST_RSP_COMMAND, *responseSize* to 00 00 00 $0A_{16}$ and *responseCode* to TPM_RC_BAD_TAG. This is the same response as the ISO/IEC 11889 (first edition) fatal error for TPM_BADTAG.

7.2 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 in Table 3. Another set of response code values 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 response codes that are not command specific are listed and specified 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

Table 4 — Command-Independent Response Codes

Response Code	Meaning
TPM_RC_CANCELED	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 exeucte 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 explicity 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 commannd. 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 1st handle and 6 representing the 7th. Upper values are provided for future use. The TPM resource manager needs to find the correct object and load it. It may then adjust the handle and retry the command.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

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 1st session handle and 6 representing the 7th. Upper values are provided for future use. 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 respose 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_SUCCESS, 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 ISO/IEC 11889-1, clause 38, “Multi-tasking”

NOTE 1 Usually, the TPM_RC_REFERENCE_Hx and TPM_RC_REFERENCE_Sx response codes indicate the TPM resource manager has a corrupted database.

NOTE 2 The TPM_RC_YIELDED response code won’t occur in the reference implementation.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

8 Implementation Dependent

The actions code for each command makes assumptions about the behavior of various sub-systems. There are many possible implementations of the subsystems that would achieve 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

9 Detailed Actions Assumptions

9.1 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.

9.2 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 needs 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 command parameters. If the response has handles or parameters, the calling stack contains a pointer to a data structure (**out**) to hold the handles and response 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 ISO/IEC 11889-2.
- Space set aside for the **out** structure is sufficient to hold the largest **out** structure that could be produced by the command

9.3 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

10 Start-up

10.1 Introduction

Clause 10 contains the commands used to manage the startup and restart state of a TPM.

10.2 _TPM_Init

10.2.1 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 might 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

10.2.2 Detailed Actions

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

```
#include "InternalRoutines.h"
LIB_EXPORT void
_TPM_Init(
    void
    )
{
    // Clear the failure mode flags
    g_inFailureMode = FALSE;
    g_forceFailureMode = FALSE;

    // Initialize the NvEnvironment.
    g_nvOk = NvPowerOn();

    // Initialize crypto engine
    CryptInitUnits();

    // Start clock
    TimePowerOn();

    // Set initialization state
    TPMInit();

    // Initialize object table
    ObjectStartup();

    // Set g_DRTMHandle as unassigned
    g_DRTMHandle = TPM_RH_UNASSIGNED;

    // No H-CRTM, yet.
    g_DrtmPreStartup = FALSE;


    return;
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

10.3 TPM2_Startup

10.3.1 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 10.2.1 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 might 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* and *phEnableNV*.

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* and *phEnableNV* 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

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

- *platformAuth* and *platformPolicy* shall be set to the Empty Buffer,
- For each NV index with TPMA_NV_WRITE_DEFINE CLEAR or TPMA_NV_WRITTEN CLEAR, TPMA_NV_WRITELOCKED shall be CLEAR,
- For each NV index with TPMA_NV_CLEAR_STCLEAR SET, TPMA_NV_WRITTEN shall be CLEAR,
- 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 and the H-CRTM state (see ISO/IEC 11889-1, *clause 34.3, "H-CRTM before TPM2_Startup()"*)

NOTE 4 PCR can 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 ISO/IEC 11889-1, clause 17.1, "Initializing PCR" in 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,
- For each NV index with TPMA_NV_WRITE_DEFINE CLEAR or TPMA_NV_WRITTEN CLEAR, TPMA_NV_WRITELOCKED shall be CLEAR,
- For each NV index with TPMA_NV_CLEAR_STCLEAR SET, TPMA_NV_WRITTEN shall be CLEAR, and
- PCR in all banks are reset to their default initial conditions.
- If an H-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 need TPM_SU_CLEAR when no shutdown was performed or after Shutdown(CLEAR).

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

10.3.2 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 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

10.3.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Startup_fp.h"
#ifdef TPM_CC_Startup  // Conditional expansion of this file
```

Table 7 — TPM2_Startup Errors

Error Returns	Meaning
TPM_RC_LOCALITY	a Startup(STATE) does not have the same H-CRTM state as the previous Startup().
TPM_RC_NV_UNINITIALIZED	the saved state cannot be recovered and a Startup(CLEAR) is requried.
TPM_RC_VALUE	start up type is not compatible with previous shutdown sequence

```
TPM_RC
TPM2_Startup(
    Startup_In      *in             // IN: input parameter list
    )
{
    STARTUP_TYPE         startup;
    TPM_RC               result;
    BOOL                 prevDrtmPreStartup;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS)
        return result;

// Input Validation

    // Read orderly shutdown states from previous power cycle
    NvReadReserved(NV_ORDERLY, &g_prevOrderlyState);

    // HACK to extract the DRTM startup type associated with the previous shutdown
    prevDrtmPreStartup = (g_prevOrderlyState == (TPM_SU_STATE + PRE_STARTUP_FLAG));
    if(prevDrtmPreStartup)
        g_prevOrderlyState = TPM_RC_LOCALITY;

    // if this startup is a TPM Resume, then the H-CRTM states have to match
    if(     in->startupType == TPM_SU_STATE
        && g_DrtmPreStartup != prevDrtmPreStartup)
        return TPM_RC_VALUE + RC_Startup_startupType;


    // if the previous power cycle was shut down with no StateSave command, or
    // with StateSave command for CLEAR, or the part of NV used for  TPM_SU_STATE
    // cannot be recovered, then this cycle can not startup up with STATE
    if(in->startupType == TPM_SU_STATE)
    {
        if(     g_prevOrderlyState == SHUTDOWN_NONE
           ||   g_prevOrderlyState == TPM_SU_CLEAR)
            return TPM_RC_VALUE + RC_Startup_startupType;

        if(g_nvOk == FALSE)
            return TPM_RC_NV_UNINITIALIZED;
    }

// Internal Date Update
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
50
51      // Translate the TPM2_ShutDown and TPM2_Startup sequence into the startup
52      // types. Will only be a SU_RESTART if the NV is OK
53      if(     in->startupType == TPM_SU_CLEAR
54          &&  g_prevOrderlyState == TPM_SU_STATE
55          &&  g_nvOk == TRUE)
56      {
57          startup = SU_RESTART;
58          // Read state reset data
59          NvReadReserved(NV_STATE_RESET, &gr);
60      }
61      // In this check, we don't need to look at g_nvOk because that was checked
62      // above
63      else if(in->startupType == TPM_SU_STATE && g_prevOrderlyState == TPM_SU_STATE)
64      {
65          // For a resume, the H-CRTM startup method must be the same
66          if(g_DrtmPreStartup != prevDrtmPreStartup)
67              return TPM_RC_LOCALITY;
68
69          // Read state clear and state reset data
70          NvReadReserved(NV_STATE_CLEAR, &gc);
71          NvReadReserved(NV_STATE_RESET, &gr);
72          startup = SU_RESUME;
73      }
74      else
75      {
76          startup = SU_RESET;
77      }
78
79      // Read persistent data from NV
80      NvReadPersistent();
81
82      // Crypto Startup
83      CryptUtilStartup(startup);
84
85      // Read the platform unique value that is used as VENDOR_PERMANENT auth value
86      g_platformUniqueDetails.t.size = (UINT16)_plat__GetUnique(1,
87                                       sizeof(g_platformUniqueDetails.t.buffer),
88                                       g_platformUniqueDetails.t.buffer);
89
90      // Start up subsystems
91      // Start counters and timers
92      TimeStartup(startup);
93
94      // Start dictionary attack subsystem
95      DAStartup(startup);
96
97      // Enable hierarchies
98      HierarchyStartup(startup);
99
100     // Restore/Initialize PCR
101     PCRStartup(startup);
102
103     // Restore/Initialize command audit information
104     CommandAuditStartup(startup);
105
106     // Object context variables
107     if(startup == SU_RESET)
108     {
109         // Reset object context ID to 0
110         gr.objectContextID = 0;
111         // Reset clearCount to 0
112         gr.clearCount= 0;
113     }
114
115     // Initialize session table
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
116         SessionStartup(startup);
117
118         // Initialize index/evict data.  This function clear read/write locks
119         // in NV index
120         NvEntityStartup(startup);
121
122         // Initialize the orderly shut down flag for this cycle to SHUTDOWN_NONE.
123         gp.orderlyState = SHUTDOWN_NONE;
124         NvWriteReserved(NV_ORDERLY, &gp.orderlyState);
125
126         // Update TPM internal states if command succeeded.
127         // Record a TPM2_Startup command has been received.
128         TPMRegisterStartup();
129
130
131         // The H-CRTM state no longer matters
132         g_DrtmPreStartup = FALSE;
133
134         return TPM_RC_SUCCESS;
135
136     }
137     #endif // CC_Startup
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

10.4 TPM2_Shutdown

10.4.1 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_ORDERY 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. The TPM MAY nullify this command for any subsequent command rather than check whether the command changed state saved by this command. If this command is nullified. and if no TPM2_Shutdown() occurs before the next TPM2_Startup(), then the next TPM2_Startup() shall be TPM2_Startup(CLEAR).

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

10.4.2 Command and Response

Table 8 — TPM2_Shutdown Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_Shutdown {NV}
TPM_SU	shutdownType	TPM_SU_CLEAR or TPM_SU_STATE

Table 9 — TPM2_Shutdown Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

10.4.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Shutdown_fp.h"
#ifdef TPM_CC_Shutdown  // Conditional expansion of this file
```

Table 10 — TPM2_Shutdown Errors

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

```
TPM_RC
TPM2_Shutdown(
    Shutdown_In     *in             // IN: input parameter list
    )
{
    TPM_RC           result;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

// Input Validation

    // If PCR bank has been reconfigured, a CLEAR state save is required
    if(g_pcrReConfig && in->shutdownType == TPM_SU_STATE)
        return TPM_RC_TYPE + RC_Shutdown_shutdownType;

// Internal Data Update

    // PCR private date state save
    PCRStateSave(in->shutdownType);

    // Get DRBG state
    CryptDrbgGetPutState(GET_STATE);

    // Save all orderly data.
    NvWriteReserved(NV_ORDERLY_DATA, &go);

    // Save RAM backed NV index data
    NvStateSave();

    if(in->shutdownType == TPM_SU_STATE)
    {
        // Save STATE_RESET and STATE_CLEAR data
        NvWriteReserved(NV_STATE_CLEAR, &gc);
        NvWriteReserved(NV_STATE_RESET, &gr);
    }
    else if(in->shutdownType == TPM_SU_CLEAR)
    {
        // Save STATE_RESET data
        NvWriteReserved(NV_STATE_RESET, &gr);
    }

    // Write orderly shut down state
    if(in->shutdownType == TPM_SU_CLEAR)
        gp.orderlyState = TPM_SU_CLEAR;
    else if(in->shutdownType == TPM_SU_STATE)
        gp.orderlyState = TPM_SU_STATE
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
54                                  | ((g_DrtmPreStartup) ? PRE_STARTUP_FLAG : 0);
55      else
56          pAssert(FALSE);
57
58      NvWriteReserved(NV_ORDERLY, &gp.orderlyState);
59
60      // If PRE_STARTUP_FLAG was SET, then it will stay set in gp.orderlyState even
61      // if the TPM isn't actually shut down. This is OK because all other checks
62      // of gp.orderlyState are to see if it is SHUTDOWN_NONE. So, having
63      // gp.orderlyState set to another value that is also not SHUTDOWN_NONE, is not
64      // an issue. This must be the case, otherwise, it would be impossible to add
65      // an additional shutdown type without major changes to the code.
66
67      return TPM_RC_SUCCESS;
68  }
69  #endif // CC_Shutdown
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

11 Testing

11.1 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() can be executed before the hash algorithms have been tested. However, until the hash algorithms have been tested, the contents of a PCR cannot be used in any command if that command could 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 can 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

11.2 TPM2_SelfTest

11.2.1 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 returns 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 might 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 can 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 might be available for signaling normal command completion. It is not necessary that there be two interrupts, but the interface ought to provide a way to indicate the nature of the interrupt (normal command or deferred command).

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

11.2.2 Command and Response

Table 11 — TPM2_SelfTest Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
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 12 — TPM2_SelfTest Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

11.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "SelfTest_fp.h"
#ifdef TPM_CC_SelfTest  // Conditional expansion of this file
```

Table 13 — TPM2_SelfTest Errors

Error Returns	Meaning
TPM_RC_CANCELED	the command was canceled (some incremental process may have been made)
TPM_RC_TESTING	self test in process

```
TPM_RC
TPM2_SelfTest(
    SelfTest_In     *in             // IN: input parameter list
    )
{
// Command Output

    // Call self test function in crypt module
    return CryptSelfTest(in->fullTest);
}
#endif // CC_SelfTest
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

11.3 TPM2_IncrementalSelfTest

11.3.1 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.

The implementation can treat algorithms on the *toTest* list as either 'test each completely' or 'test this combination.'

EXAMPLE 1 If the *toTest* list includes AES and CTR mode, it can be interpreted as a request to test only AES in CTR mode. Alternatively, it can be interpreted as a request to test AES in all modes and CTR mode for all symmetric algorithms.

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.

NOTE 3 An algorithm remains on the *toDoList* while any part of it remains untested.

EXAMPLE 2 A symmetric algorithm remains untested until it is tested with all its modes.

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

NOTE 5 If none of the algorithms listed in *toTest* is in the *toDoList*, then no tests will be performed.

NOTE 6 The TPM cannot return TPM_RC_TESTING for this command, even when testing is not complete, because response parameters can only returned with the TPM_RC_SUCCESS return code.

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

NOTE 7 An implementation might perform all requested tests before returning TPM_RC_SUCCESS, or it might return TPM_RC_SUCCESS for this command and then return TPM_RC_TESTING for all subsequence commands (including TPM2_IncrementatSelfTest()) until the requested tests are complete.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

11.3.2 Command and Response

Table 14 — TPM2_IncrementalSelfTest Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_IncrementalSelfTest {NV}
TPML_ALG	toTest	list of algorithms that should be tested

Table 15 — TPM2_IncrementalSelfTest Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPML_ALG	toDoList	list of algorithms that need testing

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

11.3.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "IncrementalSelfTest_fp.h"
#ifdef TPM_CC_IncrementalSelfTest  // Conditional expansion of this file
```

Table 16 — TPM2_IncrementalSelfTest Errors

Error Returns	Meaning
TPM_RC_CANCELED	the command was canceled (some tests may have completed)
TPM_RC_VALUE	an algorithm in the *toTest* list is not implemented

```
TPM_RC
TPM2_IncrementalSelfTest(
    IncrementalSelfTest_In      *in,            // IN: input parameter list
    IncrementalSelfTest_Out     *out            // OUT: output parameter list
    )
{
    TPM_RC                       result;
// Command Output

    // Call incremental self test function in crypt module. If this function
    // returns TPM_RC_VALUE, it means that an algorithm on the 'toTest' list is
    // not implemented.
    result = CryptIncrementalSelfTest(&in->toTest, &out->toDoList);
    if(result == TPM_RC_VALUE)
        return TPM_RCS_VALUE + RC_IncrementalSelfTest_toTest;
    return result;
}
#endif // CC_IncrementalSelfTest
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

11.4 TPM2_GetTestResult

11.4.1 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.

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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

11.4.2 Command and Response

Table 17 — TPM2_GetTestResult Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_GetTestResult

Table 18 — TPM2_GetTestResult Response

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_MAX_BUFFER	outData	test result data contains manufacturer-specific information
TPM_RC	testResult	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

11.4.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "GetTestResult_fp.h"
#ifdef TPM_CC_GetTestResult  // Conditional expansion of this file
```

In the reference implementation, this function is only reachable if the TPM is not in failure mode meaning that all tests that have been run have completed successfully. There is not test data and the test result is TPM_RC_SUCCESS.

```
TPM_RC
TPM2_GetTestResult(
    GetTestResult_Out   *out            // OUT: output parameter list
    )
{
// Command Output

    // Call incremental self test function in crypt module
    out->testResult = CryptGetTestResult(&out->outData);

    return TPM_RC_SUCCESS;
}
#endif // CC_GetTestResult
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

12 Session Commands

12.1 TPM2_StartAuthSession

12.1.1 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* needs to be an Empty Buffer.

The label value of “SECRET” (see ISO/IEC 11889-1, clause 5.4, "KDF Label Parameters" for normative KDF label values) 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 *bind* parameter contributes an authorization value to the *sessionKey* generation process.

If both *tpmKey* and *bind* are TPM_ALG_NULL, then *sessionKey* is set to the Empty Buffer. If *tpmKey* is not TPM_ALG_NULL, then *encryptedSalt* is used in the computation of *sessionKey*. If *bind* is not TPM_ALG_NULL, the *authValue* of *bind* 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 *symmetric* 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 ISO/IEC 11889-1, clause 30, “Context Management”).

If *tpmKey* is not TPM_ALG_NULL then *encryptedSalt* 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

   1) *encryptedSalt* does not contain a value that is the size of the public modulus of *tpmKey*,

   2) *encryptedSalt* has a value that is greater than the public modulus of *tpmKey*,

   3) *encryptedSalt* is not a properly encoded OAEP value, or

   4) the decrypted *salt* value is larger than the size of the digest produced by the *nameAlg* of *tpmKey*; or

b) *tpmKey* references an ECC key and *encryptedSalt*

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

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 needed 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*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

12.1.2 Command and Response

Table 19 — TPM2_StartAuthSession Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit, decrypt, or encrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_StartAuthSession
TPMI_DH_OBJECT+	tpmKey	handle of a loaded decrypt key used to encrypt *salt* may be TPM_RH_NULL Auth Index: None
TPMI_DH_ENTITY+	bind	entity providing the *authValue* may be TPM_RH_NULL Auth Index: None
TPM2B_NONCE	nonceCaller	initial *nonceCaller,* sets nonce size for the session shall be at least 16 octets
TPM2B_ENCRYPTED_SECRET	encryptedSalt	value encrypted according to the type of *tpmKey* If *tpmKey* 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 20 — TPM2_StartAuthSession Response

Type	Name	Description
TPM_ST	tag	see clause 7
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 *sessionKey*

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

12.1.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "StartAuthSession_fp.h"
#ifdef TPM_CC_StartAuthSession  // Conditional expansion of this file
```

Table 21 — TPM2_StartAuthSession Errors

Error Returns	Meaning
TPM_RC_ATTRIBUTES	*tpmKey* 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
TPM_RC_MODE	*symmetric* 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 *authHash*
TPM_RC_VALUE	secret size does not match decrypt key type; or the recovered secret is larger than the digest size of the *nameAlg* of *tpmKey*; or, for an RSA decrypt key, if *encryptedSecret* is greater than the public exponent of *tpmKey*.

```
TPM_RC
TPM2_StartAuthSession(
    StartAuthSession_In     *in,            // IN: input parameter buffer
    StartAuthSession_Out    *out            // OUT: output parameter buffer
    )
{
    TPM_RC                   result = TPM_RC_SUCCESS;
    OBJECT                  *tpmKey;                // TPM key for decrypt salt
    SESSION                 *session;               // session internal data
    TPM2B_DATA               salt;

// Input Validation

    // Check input nonce size.  IT should be at least 16 bytes but not larger
    // than the digest size of session hash.
    if(    in->nonceCaller.t.size < 16
        || in->nonceCaller.t.size > CryptGetHashDigestSize(in->authHash))
        return TPM_RC_SIZE + RC_StartAuthSession_nonceCaller;

    // If an decrypt key is passed in, check its validation
    if(in->tpmKey != TPM_RH_NULL)
    {
        // secret size cannot be 0
        if(in->encryptedSalt.t.size == 0)
            return TPM_RC_VALUE + RC_StartAuthSession_encryptedSalt;

        // Get pointer to loaded decrypt key
        tpmKey = ObjectGet(in->tpmKey);

        // Decrypting salt requires accessing the private portion of a key.
        // Therefore, tmpKey can not be a key with only public portion loaded
        if(tpmKey->attributes.publicOnly)
            return TPM_RC_HANDLE + RC_StartAuthSession_tpmKey;
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
37
38          // HMAC session input handle check.
39          // tpmKey should be a decryption key
40          if(tpmKey->publicArea.objectAttributes.decrypt != SET)
41              return TPM_RC_ATTRIBUTES + RC_StartAuthSession_tpmKey;
42
43          // Secret Decryption.  A TPM_RC_VALUE, TPM_RC_KEY or Unmarshal errors
44          // may be returned at this point
45          // See ISO/IEC 11889-1, clause 5.4, "KDF Label Parameters"
46          result = CryptSecretDecrypt(in->tpmKey, &in->nonceCaller, "SECRET",
47                                      &in->encryptedSalt, &salt);
48          if(result != TPM_RC_SUCCESS)
49              return TPM_RC_VALUE + RC_StartAuthSession_encryptedSalt;
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  }
91  #endif // CC_StartAuthSession
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

12.2 TPM2_PolicyRestart

12.2.1 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 because the session restarts with the same *nonceTPM*. 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

12.2.2 Command and Response

Table 22 — TPM2_PolicyRestart Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyRestart
TPMI_SH_POLICY	sessionHandle	the handle for the policy session

Table 23 — TPM2_PolicyRestart Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

12.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyRestart_fp.h"
#ifdef TPM_CC_PolicyRestart  // Conditional expansion of this file
TPM_RC
TPM2_PolicyRestart(
    PolicyRestart_In    *in             // IN: input parameter list
    )
{
    SESSION                 *session;
    BOOL                     wasTrialSession;

// Internal Data Update

    session = SessionGet(in->sessionHandle);
    wasTrialSession = session->attributes.isTrialPolicy == SET;

    // Initialize policy session
    SessionResetPolicyData(session);

    session->attributes.isTrialPolicy = wasTrialSession;

    return TPM_RC_SUCCESS;
}
#endif // CC_PolicyRestart
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13 Object Commands

13.1 TPM2_Create

13.1.1 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 ISO/IEC 11889-1, clause 27.2, "Public Area Template" and in ISO/IEC 11889-2, clause 9.3, "TPMA_OBJECT".

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.userAuth),* 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 ISO/IEC 11889-2, clause 9.3, "TPMA_OBJECT".

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

The methods in clause 13.1 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 ISO/IEC 11889-1.

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*.sensitive.*data* is the Empty Buffer, a TPM-generated key value is placed in the new object's TPMT_SENSITIVE.*sensitive.sym*. The size of the key will be determined by *inPublic.publicArea.parameters*.

   2) If *inSensitive*.sensitive.*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.*sensitive*.*sym* of the new object.

   3) A TPM-generated obfuscation value is placed in TPMT_SENSITIVE.*sensitive.seedValue*. The size of the obfuscation value is the size of the digest produced by the nameAlg in *inPublic*. This value prevents the public *unique* value from leaking information about the *sensitive* area.

   4) The TPMT_PUBLIC.*unique.sym* 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.

$$unique := \mathbf{H}_{nameAlg}(sensitive.seedValue.buffer \,||\, sensitive.any.buffer) \quad (1)$$

b) If the Object is an asymmetric key:

   1) If *inSensitive.sensitive.data* is not the Empty Buffer, then the TPM shall return TPM_RC_VALUE.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

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.*seedValue* value is created; otherwise, TPMT_SENSITIVE.*seedValue.size* is set to zero.

NOTE 1 An Object that is not a storage key has no child Objects to encrypt, so it does not need a symmetric key.

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 pubic area of the template, then the TPM shall return TPM_RC_KDF.

NOTE 2 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 needs 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.sensitive.data* is an Empty Buffer, and neither *sign* nor *decrypt* is SET in *inPublic.attributes*, the TPM shall return TPM_RC_ATTRIBUTES. This would be a data object with no data.

2) If *inSensitive.sensitive.data* is not an Empty Buffer, the TPM will copy the *inSensitive.sensitive.data* to TPMT_SENSITIVE.*sensitive,bits* of the new object.

NOTE 3 The size of *inSensitive.sensitive.data* is limited to be no larger than the largest value of TPMT_SENSITIVE.*sensitive.bits* by MAX_SYM_DATA.

3) If *inSensitive.sensitive.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.bits*.

4) A TPM-generated obfuscation value that is the size of the digest produced by the *nameAlg* of *inPublic* is placed in TPMT_SENSITIVE.*seedValue*.

5) The TPMT_PUBLIC.*unique.keyedHash* 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 4 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. TPMS_CREATION_DATA.*outsideInfo* is set to *outsideInfo*. 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 and parameters 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*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

EXAMPLE If the *inPublic.parameters.ecc.symmetric.algorithm* does not match the parent, the TPM needs to return TPM_RC_ ASYMMETRIC rather than TPM_RC_SYMMETRIC.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.1.2 Command and Response

Table 24 — TPM2_Create Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 25 — TPM2_Create Response

Type	Name	Description
TPM_ST	tag	see clause 7
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 *creationData* using *nameAlg* of *outPublic*
TPMT_TK_CREATION	creationTicket	ticket used by TPM2_CertifyCreation() to validate that the creation data was produced by the TPM

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.1.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Object_spt_fp.h"
#include "Create_fp.h"
#ifdef TPM_CC_Create  // Conditional expansion of this file
```

Table 26 — TPM2_Create Errors

Error Returns	Meaning
TPM_RC_ASYMMETRIC	non-duplicable storage key and its parent have different public parameters
TPM_RC_ATTRIBUTES	*sensitiveDataOrigin* is CLEAR when 'sensitive. data' is an Empty Buffer, or is SET when 'sensitive. data' is not empty; *fixedTPM*, *fixedParent*, or *encryptedDuplication* attributes are inconsistent between themselves or with those of the parent object; inconsistent *restricted*, *decrypt* and *sign* 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_RANGE	FOr() an RSA key, the exponent value is not supported.
TPM_RC_SCHEME	inconsistent attributes *decrypt*, *sign*, *restricted* 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; *parentHandle* 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

```
TPM_RC
TPM2_Create(
    Create_In       *in,            // IN: input parameter list
    Create_Out      *out            // OUT: output parameter list
    )
{
    TPM_RC                  result = TPM_RC_SUCCESS;
    TPMT_SENSITIVE          sensitive;
    TPM2B_NAME              name;
```

© ISO/IEC 2015 – All rights reserved

```
14
15  // Input Validation
16
17      OBJECT      *parentObject;
18
19      parentObject = ObjectGet(in->parentHandle);
20
21      // Does parent have the proper attributes?
22      if(!AreAttributesForParent(parentObject))
23          return TPM_RC_TYPE + RC_Create_parentHandle;
24
25      // The sensitiveDataOrigin attribute must be consistent with the setting of
26      // the size of the data object in inSensitive.
27      if(   (in->inPublic.t.publicArea.objectAttributes.sensitiveDataOrigin == SET)
28         != (in->inSensitive.t.sensitive.data.t.size == 0))
29          // Mismatch between the object attributes and the parameter.
30          return TPM_RC_ATTRIBUTES + RC_Create_inSensitive;
31
32      // Check attributes in input public area. TPM_RC_ASYMMETRIC, TPM_RC_ATTRIBUTES,
33      // TPM_RC_HASH, TPM_RC_KDF, TPM_RC_SCHEME, TPM_RC_SIZE, TPM_RC_SYMMETRIC,
34      // or TPM_RC_TYPE error may be returned at this point.
35      result = PublicAttributesValidation(FALSE, in->parentHandle,
36                                          &in->inPublic.t.publicArea);
37      if(result != TPM_RC_SUCCESS)
38          return RcSafeAddToResult(result, RC_Create_inPublic);
39
40      // Validate the sensitive area values
41      if(  MemoryRemoveTrailingZeros(&in->inSensitive.t.sensitive.userAuth)
42             > CryptGetHashDigestSize(in->inPublic.t.publicArea.nameAlg))
43          return TPM_RC_SIZE + RC_Create_inSensitive;
44
45  // Command Output
46
47      // Create object crypto data
48      result = CryptCreateObject(in->parentHandle, &in->inPublic.t.publicArea,
49                                 &in->inSensitive.t.sensitive, &sensitive);
50      if(result != TPM_RC_SUCCESS)
51          return result;
52
53      // Fill in creation data
54      FillInCreationData(in->parentHandle, in->inPublic.t.publicArea.nameAlg,
55                         &in->creationPCR, &in->outsideInfo,
56                         &out->creationData, &out->creationHash);
57
58      // Copy public area from input to output
59      out->outPublic.t.publicArea = in->inPublic.t.publicArea;
60
61      // Compute name from public area
62      ObjectComputeName(&(out->outPublic.t.publicArea), &name);
63
64      // Compute creation ticket
65      TicketComputeCreation(EntityGetHierarchy(in->parentHandle), &name,
66                            &out->creationHash, &out->creationTicket);
67
68      // Prepare output private data from sensitive
69      SensitiveToPrivate(&sensitive, &name, in->parentHandle,
70                         out->outPublic.t.publicArea.nameAlg,
71                         &out->outPrivate);
72
73      return TPM_RC_SUCCESS;
74  }
75  #endif // CC_Create
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.2 TPM2_Load

13.2.1 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 to be 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 attributes will be checked according to the rules defined in in ISO/IEC 11889-2, clause 9.3, "TPMA_OBJECT".

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 needs to 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 needs to be zero and it needs to 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 needs to be f(x) where x is the private key.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

13.2.2 Command and Response

Table 27 — TPM2_Load Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 28 — TPM2_Load Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM_HANDLE	objectHandle	handle for the loaded object
TPM2B_NAME	name	Name of the loaded object

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Load_fp.h"
#ifdef TPM_CC_Load  // Conditional expansion of this file
#include "Object_spt_fp.h"
```

Table 29 — TPM2_Load Errors

Error Returns	Meaning
TPM_RC_ASYMMETRIC	storage key with different asymmetric type than parent
TPM_RC_ATTRIBUTES	*inPulblic* attributes are not allowed with selected parent
TPM_RC_BINDING	*inPrivate* and *inPublic* are not cryptographically bound
TPM_RC_HASH	incorrect hash selection for signing key
TPM_RC_INTEGRITY	HMAC on *inPrivate* was not valid
TPM_RC_KDF	KDF selection not allowed
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	no available object slot
TPM_RC_SCHEME	the signing scheme is not valid for the key
TPM_RC_SENSITIVE	the *inPrivate* did not unmarshal correctly
TPM_RC_SIZE	*inPrivate* missing, or *authPolicy* size for *inPublic* or is not valid
TPM_RC_SYMMETRIC	symmetric algorithm not provided when required
TPM_RC_TYPE	*parentHandle* 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

```
TPM_RC
TPM2_Load(
    Load_In         *in,            // IN: input parameter list
    Load_Out        *out            // OUT: output parameter list
    )
{
    TPM_RC                  result = TPM_RC_SUCCESS;
    TPMT_SENSITIVE          sensitive;
    TPMI_RH_HIERARCHY       hierarchy;
    OBJECT                 *parentObject = NULL;
    BOOL                    skipChecks = FALSE;

// Input Validation
    if(in->inPrivate.t.size == 0)
        return TPM_RC_SIZE + RC_Load_inPrivate;

    parentObject = ObjectGet(in->parentHandle);
    // Is the object that is being used as the parent actually a parent.
    if(!AreAttributesForParent(parentObject))
        return TPM_RC_TYPE + RC_Load_parentHandle;

    // If the parent is fixedTPM, then the attributes of the object
    // are either "correct by construction" or were validated
    // when the object was imported.  If they pass the integrity
    // check, then the values are valid
    if(parentObject->publicArea.objectAttributes.fixedTPM)
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
31          skipChecks = TRUE;
32      else
33      {
34          // If parent doesn't have fixedTPM SET, then this can't have
35          // fixedTPM SET.
36          if(in->inPublic.t.publicArea.objectAttributes.fixedTPM == SET)
37              return TPM_RC_ATTRIBUTES + RC_Load_inPublic;
38
39          // Perform self check on input public area.  A TPM_RC_SIZE, TPM_RC_SCHEME,
40          // TPM_RC_VALUE, TPM_RC_SYMMETRIC, TPM_RC_TYPE, TPM_RC_HASH,
41          // TPM_RC_ASYMMETRIC, TPM_RC_ATTRIBUTES or TPM_RC_KDF error may be returned
42          // at this point
43          result = PublicAttributesValidation(TRUE, in->parentHandle,
44                                              &in->inPublic.t.publicArea);
45          if(result != TPM_RC_SUCCESS)
46              return RcSafeAddToResult(result, RC_Load_inPublic);
47      }
48
49      // Compute the name of object
50      ObjectComputeName(&in->inPublic.t.publicArea, &out->name);
51
52      // Retrieve sensitive data.  PrivateToSensitive() may return TPM_RC_INTEGRITY or
53      // TPM_RC_SENSITIVE
54      // errors may be returned at this point
55      result = PrivateToSensitive(&in->inPrivate, &out->name, in->parentHandle,
56                                  in->inPublic.t.publicArea.nameAlg,
57                                  &sensitive);
58      if(result != TPM_RC_SUCCESS)
59          return RcSafeAddToResult(result, RC_Load_inPrivate);
60
61  // Internal Data Update
62
63      // Get hierarchy of parent
64      hierarchy = ObjectGetHierarchy(in->parentHandle);
65
66      // Create internal object.  A lot of different errors may be returned by this
67      // loading operation as it will do several validations, including the public
68      // binding check
69      result = ObjectLoad(hierarchy, &in->inPublic.t.publicArea, &sensitive,
70                          &out->name, in->parentHandle, skipChecks,
71                          &out->objectHandle);
72
73      if(result != TPM_RC_SUCCESS)
74          return result;
75
76      return TPM_RC_SUCCESS;
77  }
78  #endif // CC_Load
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.3 TPM2_LoadExternal

13.3.1 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 so the TPM can 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 cannot appear to be part of a hierarchy.

The object's TPMA_OBJECT attributes will be checked according to the rules defined in in ISO/IEC 11889-2, clause 9.3, "TPMA_OBJECT". 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 might 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*, since, its private area would not be available in the clear 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 ought to 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 pubic 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

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, the object is part of no hierarchy, and there is no implicit flush.

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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

13.3.2 Command and Response

Table 30 — TPM2_LoadExternal Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit, encrypt, or derypt session is present; otherwise, TPM_ST_NO_SESSIONS
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 31 — TPM2_LoadExternal Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM_HANDLE	objectHandle	handle for the loaded object
TPM2B_NAME	name	name of the loaded object

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.3.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "LoadExternal_fp.h"
#ifdef TPM_CC_LoadExternal  // Conditional expansion of this file
#include "Object_spt_fp.h"
```

Table 32 — TPM2_LoadExternal Errors

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

```
TPM_RC
TPM2_LoadExternal(
    LoadExternal_In     *in,            // IN: input parameter list
    LoadExternal_Out    *out            // OUT: output parameter list
    )
{
    TPM_RC               result;
    TPMT_SENSITIVE      *sensitive;
    BOOL                 skipChecks;

// Input Validation

    // If the target hierarchy is turned off, the object can not be loaded.
    if(!HierarchyIsEnabled(in->hierarchy))
        return TPM_RC_HIERARCHY + RC_LoadExternal_hierarchy;

    // the size of authPolicy is either 0 or the digest size of nameAlg
    if(in->inPublic.t.publicArea.authPolicy.t.size != 0
            && in->inPublic.t.publicArea.authPolicy.t.size !=
            CryptGetHashDigestSize(in->inPublic.t.publicArea.nameAlg))
        return TPM_RC_SIZE + RC_LoadExternal_inPublic;

    // For loading an object with both public and sensitive
    if(in->inPrivate.t.size != 0)
    {
        // An external object can only be loaded at TPM_RH_NULL hierarchy
        if(in->hierarchy != TPM_RH_NULL)
            return TPM_RC_HIERARCHY + RC_LoadExternal_hierarchy;
        // An external object with a sensitive area must have fixedTPM == CLEAR
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
34          // fixedParent == CLEAR, and must have restrict CLEAR so that it does not
35          // appear to be a key that was created by this TPM.
36          if(   in->inPublic.t.publicArea.objectAttributes.fixedTPM != CLEAR
37             || in->inPublic.t.publicArea.objectAttributes.fixedParent != CLEAR
38             || in->inPublic.t.publicArea.objectAttributes.restricted != CLEAR
39            )
40              return TPM_RC_ATTRIBUTES + RC_LoadExternal_inPublic;
41      }
42
43      // Validate the scheme parameters
44      result = SchemeChecks(TRUE, TPM_RH_NULL, &in->inPublic.t.publicArea);
45      if(result != TPM_RC_SUCCESS)
46              return RcSafeAddToResult(result, RC_LoadExternal_inPublic);
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  }
66  #endif // CC_LoadExternal
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.4 TPM2_ReadPublic

13.4.1 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 object, the TPM shall return TPM_RC_SEQUENCE.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

13.4.2 Command and Response

Table 33 — TPM2_ReadPublic Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or encrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ReadPublic
TPMI_DH_OBJECT	objectHandle	TPM handle of an object Auth Index: None

Table 34 — TPM2_ReadPublic Response

Type	Name	Description
TPM_ST	tag	see clause 7
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

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.4.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "ReadPublic_fp.h"
#ifdef TPM_CC_ReadPublic  // Conditional expansion of this file
```

Table 35 — TPM2_ReadPublic Errors

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

```
TPM_RC
TPM2_ReadPublic(
    ReadPublic_In   *in,            // IN: input parameter list
    ReadPublic_Out  *out            // OUT: output parameter list
    )
{
    OBJECT                  *object;

// Input Validation

    // Get loaded object pointer
    object = ObjectGet(in->objectHandle);

    // Can not read public area of a sequence object
    if(ObjectIsSequence(object))
        return TPM_RC_SEQUENCE;

// Command Output

    // Compute size of public area in canonical form
    out->outPublic.t.size = TPMT_PUBLIC_Marshal(&object->publicArea, NULL, NULL);

    // Copy public area to output
    out->outPublic.t.publicArea = object->publicArea;

    // Copy name to output
    out->name.t.size = ObjectGetName(in->objectHandle, &out->name.t.name);

    // Copy qualified name to output
    ObjectGetQualifiedName(in->objectHandle, &out->qualifiedName);

    return TPM_RC_SUCCESS;
}
#endif // CC_ReadPublic
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.5 TPM2_ActivateCredential

13.5.1 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 seed from secret, which is the encrypted seed. The Name of the object associated with activateHandle and the recovered seed are used in a KDF to recover the symmetric key. The recovered seed (but not the Name) is used in a KDF to recover the HMAC key.

The HMAC is used to validate that the *credentialBlob* is associated with *activateHandle* and that the data in *credentialBlob* has not been modified. The linkage to the object associated with activateHandle is achieved by including the Name in the HMAC calculation.

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.5.2 Command and Response

Table 36 — TPM2_ActivateCredential Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ActivateCredential
TPMI_DH_OBJECT	@activateHandle	handle of the object associated with certificate in *credentialBlob* Auth Index: 1 Auth Role: ADMIN
TPMI_DH_OBJECT	@keyHandle	loaded key used to decrypt the TPMS_SENSITIVE in *credentialBlob* Auth Index: 2 Auth Role: USER
TPM2B_ID_OBJECT	credentialBlob	the credential
TPM2B_ENCRYPTED_SECRET	secret	*keyHandle* algorithm-dependent encrypted seed that protects *credentialBlob*

Table 37 — TPM2_ActivateCredential Response

Type	Name	Description
TPM_ST	tag	see clause 7
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 *nameAlg* associated with *keyHandle*

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.5.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "ActivateCredential_fp.h"
#ifdef TPM_CC_ActivateCredential  // Conditional expansion of this file
#include "Object_spt_fp.h"
```

Table 38 — TPM2_ActivateCredential Errors

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

```
TPM_RC
TPM2_ActivateCredential(
    ActivateCredential_In   *in,            // IN: input parameter list
    ActivateCredential_Out  *out            // OUT: output parameter list
    )
{
    TPM_RC                   result = TPM_RC_SUCCESS;
    OBJECT                  *object;        // decrypt key
    OBJECT                  *activateObject;// key associated with
    // credential
    TPM2B_DATA               data;          // credential data

// Input Validation

    // Get decrypt key pointer
    object = ObjectGet(in->keyHandle);

    // Get certificated object pointer
    activateObject = ObjectGet(in->activateHandle);

    // input decrypt key must be an asymmetric, restricted decryption key
    if(   !CryptIsAsymAlgorithm(object->publicArea.type)
       || object->publicArea.objectAttributes.decrypt == CLEAR
       || object->publicArea.objectAttributes.restricted == CLEAR)
        return TPM_RC_TYPE + RC_ActivateCredential_keyHandle;

// Command output

    // Decrypt input credential data via asymmetric decryption.  A
    // TPM_RC_VALUE, TPM_RC_KEY or unmarshal errors may be returned at this
    // point
    // See ISO/IEC 11889-1, clause 5.4, "KDF Label Parameters"
    result = CryptSecretDecrypt(in->keyHandle, NULL,
                                "IDENTITY", &in->secret, &data);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_KEY)
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
42              return TPM_RC_FAILURE;
43          return RcSafeAddToResult(result, RC_ActivateCredential_secret);
44      }
45
46      // Retrieve secret data.  A TPM_RC_INTEGRITY error or unmarshal
47      // errors may be returned at this point
48      result = CredentialToSecret(&in->credentialBlob,
49                                  &activateObject->name,
50                                  (TPM2B_SEED *) &data,
51                                  in->keyHandle,
52                                  &out->certInfo);
53      if(result != TPM_RC_SUCCESS)
54          return RcSafeAddToResult(result,RC_ActivateCredential_credentialBlob);
55
56      return TPM_RC_SUCCESS;
57  }
58  #endif // CC_ActivateCredential
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.6 TPM2_MakeCredential

13.6.1 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 ISO/IEC 11889-1, clause 24, “Credential Protection” .

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.

This command does not use any TPM secrets nor does it require authorization. It is a convenience function, using the TPM to perform cryptographic calculations that could be done externally.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.6.2 Command and Response

Table 39 — TPM2_MakeCredential Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit, encrypt, or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
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 40 — TPM2_MakeCredential Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ID_OBJECT	credentialBlob	the credential
TPM2B_ENCRYPTED_SECRET	secret	*handle* algorithm-dependent data that wraps the key that encrypts *credentialBlob*

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.6.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "MakeCredential_fp.h"
#ifdef TPM_CC_MakeCredential  // Conditional expansion of this file
#include "Object_spt_fp.h"
```

Table 41 — TPM2_MakeCredential Errors

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

```
TPM_RC
TPM2_MakeCredential(
    MakeCredential_In   *in,            // IN: input parameter list
    MakeCredential_Out  *out            // OUT: output parameter list
    )
{
    TPM_RC                  result = TPM_RC_SUCCESS;

    OBJECT                  *object;
    TPM2B_DATA              data;

// Input Validation

    // Get object pointer
    object = ObjectGet(in->handle);

    // input key must be an asymmetric, restricted decryption key
    // NOTE: Needs to be restricted to have a symmetric value.
    if(   !CryptIsAsymAlgorithm(object->publicArea.type)
       || object->publicArea.objectAttributes.decrypt == CLEAR
       || object->publicArea.objectAttributes.restricted == CLEAR
      )
        return TPM_RC_TYPE + RC_MakeCredential_handle;

    // The credential information may not be larger than the digest size used for
    // the Name of the key associated with handle.
    if(in->credential.t.size > CryptGetHashDigestSize(object->publicArea.nameAlg))
        return TPM_RC_SIZE + RC_MakeCredential_credential;

// Command Output

    // Make encrypt key and its associated secret structure.
    // Even though CrypeSecretEncrypt() may return
    // See ISO/IEC 11889-1, clause 5.4, "KDF Label Parameters"
    out->secret.t.size = sizeof(out->secret.t.secret);
    result = CryptSecretEncrypt(in->handle, "IDENTITY", &data, &out->secret);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Prepare output credential data from secret
    SecretToCredential(&in->credential, &in->objectName, (TPM2B_SEED *) &data,
                       in->handle, &out->credentialBlob);

    return TPM_RC_SUCCESS;
}
#endif // CC_MakeCredential
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.7 TPM2_Unseal

13.7.1 General Description

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

NOTE A random, TPM-generated, Sealed Data Object can 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

13.7.2 Command and Response

Table 42 — TPM2_Unseal Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	Tag	TPM_ST_SESSIONS
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 43 — TPM2_Unseal Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_SENSITIVE_DATA	outData	unsealed data Size of *outData* is limited to be no more than 128 octets.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.7.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Unseal_fp.h"
#ifdef TPM_CC_Unseal  // Conditional expansion of this file
```

Table 44 — TPM2_Unseal Errors

Error Returns	Meaning
TPM_RC_ATTRIBUTES	*itemHandle* has wrong attributes
TPM_RC_TYPE	*itemHandle* is not a KEYEDHASH data object

```
TPM_RC
TPM2_Unseal(
    Unseal_In *in, Unseal_Out   *out
    )
{
    OBJECT                  *object;

// Input Validation

    // Get pointer to loaded object
    object = ObjectGet(in->itemHandle);

    // Input handle must be a data object
    if(object->publicArea.type != TPM_ALG_KEYEDHASH)
        return TPM_RC_TYPE + RC_Unseal_itemHandle;
    if(   object->publicArea.objectAttributes.decrypt == SET
       || object->publicArea.objectAttributes.sign == SET
       || object->publicArea.objectAttributes.restricted == SET)
        return TPM_RC_ATTRIBUTES + RC_Unseal_itemHandle;

// Command Output

    // Copy data
    MemoryCopy2B(&out->outData.b, &object->sensitive.sensitive.bits.b,
                 sizeof(out->outData.t.buffer));

    return TPM_RC_SUCCESS;
}
#endif // CC_Unseal
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.8 TPM2_ObjectChangeAuth

13.8.1 General Description

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

If successful, a new private area for the TPM-resident object associated with *objectHandle* is returned, which includes the new authorization value.

This command does not change the authorization of the TPM-resident object on which it operates. Therefore, the old authValue (of the TPM-resident object) is used when generating the response HMAC key if required..

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

NOTE 2 The TPM-resident object might 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()).

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.8.2 Command and Response

Table 45 — TPM2_ObjectChangeAuth Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 value

Table 46 — TPM2_ObjectChangeAuth Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PRIVATE	outPrivate	private area containing the new authorization value

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

13.8.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "ObjectChangeAuth_fp.h"
#ifdef TPM_CC_ObjectChangeAuth  // Conditional expansion of this file
#include "Object_spt_fp.h"
```

Table 47 — TPM2_ObjectChangeAuth Errors

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

```
TPM_RC
TPM2_ObjectChangeAuth(
    ObjectChangeAuth_In     *in,            // IN: input parameter list
    ObjectChangeAuth_Out    *out            // OUT: output parameter list
    )
{
    TPMT_SENSITIVE           sensitive;

    OBJECT                  *object;
    TPM2B_NAME               objectQN, QNCompare;
    TPM2B_NAME               parentQN;

// Input Validation

    // Get object pointer
    object = ObjectGet(in->objectHandle);

    // Can not change auth on sequence object
    if(ObjectIsSequence(object))
        return TPM_RC_TYPE + RC_ObjectChangeAuth_objectHandle;

    // Make sure that the auth value is consistent with the nameAlg
    if(  MemoryRemoveTrailingZeros(&in->newAuth)
           > CryptGetHashDigestSize(object->publicArea.nameAlg))
        return TPM_RC_SIZE + RC_ObjectChangeAuth_newAuth;

    // Check parent for object
    // parent handle must be the parent of object handle.  In this
    // implementation we verify this by checking the QN of object.  Other
    // implementation may choose different method to verify this attribute.
    ObjectGetQualifiedName(in->parentHandle, &parentQN);
    ObjectComputeQualifiedName(&parentQN, object->publicArea.nameAlg,
                               &object->name, &QNCompare);

    ObjectGetQualifiedName(in->objectHandle, &objectQN);
    if(!Memory2BEqual(&objectQN.b, &QNCompare.b))
        return TPM_RC_TYPE + RC_ObjectChangeAuth_parentHandle;

// Command Output

    // Copy internal sensitive area
    sensitive = object->sensitive;
    // Copy authValue
    sensitive.authValue = in->newAuth;

    // Prepare output private data from sensitive
    SensitiveToPrivate(&sensitive, &object->name, in->parentHandle,
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

```
52                          object->publicArea.nameAlg,
53                          &out->outPrivate);
54
55      return TPM_RC_SUCCESS;
56  }
57  #endif // CC_ObjectChangeAuth
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

14 Duplication Commands

14.1 TPM2_Duplicate

14.1.1 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 might 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. This indicates that the policy that is being used is a policy that is for duplication, and not a policy that would approve another use. That is, authority to use an object does not grant authority to duplicate the object.

The policy is likely to include cpHash in order to restrict where duplication can occur. 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

$$\mathbf{H}_{policyAlg}(objectHandle{\rightarrow}Name \,||\, newParentHandle{\rightarrow}Name) \quad (2)$$

If the compared hashes are not the same, then the TPM shall return TPM_RC_POLICY_FAIL.

NOTE 2 It is allowed that policySesion→nameHash and policySession→cpHash share the same memory space.

NOTE 3 A duplication policy is need not have either TPM2_PolicyDuplicationSelect() or TPM2_PolicyCpHash() as part of the policy. If neither is present, then the duplication policy can be satisfied with a policy that only contains TPM2_PolicyCommandCode(*code* = TPM_CC_Duplicate).

The TPM shall follow the process of encryption defined in ISO/IEC 11889-1, clause 23.3, "Duplication".

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

14.1.2 Command and Response

Table 48 — TPM2_Duplicate Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 49 — TPM2_Duplicate Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DATA	encryptionKeyOut	If the caller provided an encryption key or if *symmetricAlg* 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 *encryptionKeyIn*; and may be doubly encrypted
TPM2B_ENCRYPTED_SECRET	outSymSeed	seed protected by the asymmetric algorithms of new parent (NP)

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

14.1.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Duplicate_fp.h"
#ifdef TPM_CC_Duplicate  // Conditional expansion of this file
#include "Object_spt_fp.h"
```

Table 50 — TPM2_Duplicate Errors

Error Returns	Meaning
TPM_RC_ATTRIBUTES	key to duplicate has *fixedParent* SET
TPM_RC_HIERARCHY	*encryptedDuplication* is SET and *newParentHandle* specifies Null Hierarchy
TPM_RC_KEY	*newParentHandle* 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	*encryptedDuplication* is SET but no symmetric algorithm is provided
TPM_RC_TYPE	*newParentHandle* is neither a storage key nor TPM_RH_NULL; or the object has a NULL *nameAlg*

```
TPM_RC
TPM2_Duplicate(
    Duplicate_In    *in,            // IN: input parameter list
    Duplicate_Out   *out            // OUT: output parameter list
    )
{
    TPM_RC                  result = TPM_RC_SUCCESS;
    TPMT_SENSITIVE          sensitive;

    UINT16                  innerKeySize = 0; // encrypt key size for inner wrap

    OBJECT                  *object;
    TPM2B_DATA              data;

// Input Validation

    // Get duplicate object pointer
    object = ObjectGet(in->objectHandle);

    // duplicate key must have fixParent bit CLEAR.
    if(object->publicArea.objectAttributes.fixedParent == SET)
        return TPM_RC_ATTRIBUTES + RC_Duplicate_objectHandle;

    // Do not duplicate object with NULL nameAlg
    if(object->publicArea.nameAlg == TPM_ALG_NULL)
        return TPM_RC_TYPE + RC_Duplicate_objectHandle;

    // new parent key must be a storage object or TPM_RH_NULL
    if(in->newParentHandle != TPM_RH_NULL
            && !ObjectIsStorage(in->newParentHandle))
        return TPM_RC_TYPE + RC_Duplicate_newParentHandle;

    // If the duplicates object has encryptedDuplication SET, then there must be
    // an inner wrapper and the new parent may not be TPM_RH_NULL
    if(object->publicArea.objectAttributes.encryptedDuplication == SET)
    {
        if(in->symmetricAlg.algorithm == TPM_ALG_NULL)
            return TPM_RC_SYMMETRIC + RC_Duplicate_symmetricAlg;
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
43          if(in->newParentHandle == TPM_RH_NULL)
44              return TPM_RC_HIERARCHY + RC_Duplicate_newParentHandle;
45      }
46
47      if(in->symmetricAlg.algorithm == TPM_ALG_NULL)
48      {
49          // if algorithm is TPM_ALG_NULL, input key size must be 0
50          if(in->encryptionKeyIn.t.size != 0)
51              return TPM_RC_SIZE + RC_Duplicate_encryptionKeyIn;
52      }
53      else
54      {
55          // Get inner wrap key size
56          innerKeySize = in->symmetricAlg.keyBits.sym;
57
58          // If provided the input symmetric key must match the size of the algorithm
59          if(in->encryptionKeyIn.t.size != 0
60                  && in->encryptionKeyIn.t.size != (innerKeySize + 7) / 8)
61              return TPM_RC_SIZE + RC_Duplicate_encryptionKeyIn;
62      }
63
64  // Command Output
65
66      if(in->newParentHandle != TPM_RH_NULL)
67      {
68
69          // Make encrypt key and its associated secret structure.  A TPM_RC_KEY
70          // error may be returned at this point
71          // See ISO/IEC 11889-1, clause 5.4, "KDF Label Parameters"
72          out->outSymSeed.t.size = sizeof(out->outSymSeed.t.secret);
73          result = CryptSecretEncrypt(in->newParentHandle,
74                                      "DUPLICATE", &data, &out->outSymSeed);
75          pAssert(result != TPM_RC_VALUE);
76          if(result != TPM_RC_SUCCESS)
77              return result;
78      }
79      else
80      {
81          // Do not apply outer wrapper
82          data.t.size = 0;
83          out->outSymSeed.t.size = 0;
84      }
85
86      // Copy sensitive area
87      sensitive = object->sensitive;
88
89      // Prepare output private data from sensitive
90      SensitiveToDuplicate(&sensitive, &object->name, in->newParentHandle,
91                           object->publicArea.nameAlg, (TPM2B_SEED *) &data,
92                           &in->symmetricAlg, &in->encryptionKeyIn,
93                           &out->duplicate);
94
95      out->encryptionKeyOut = in->encryptionKeyIn;
96
97      return TPM_RC_SUCCESS;
98  }
99  #endif // CC_Duplicate
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

14.2 TPM2_Rewrap

14.2.1 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 an HMAC key and a symmetric key are recovered from *inSymSeed* 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 ISO/IEC 11889-1, clause 5.4, "KDF Label Parameters").

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.* *outSymSeed* will have a zero length. See ISO/IEC 11889-2, clause 9.3.3.9, “Bit[11] – *encryptedDuplication*”.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

14.2.2 Command and Response

Table 51 — TPM2_Rewrap Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 *inSymSeed*
TPM2B_NAME	name	the Name of the object being rewrapped
TPM2B_ENCRYPTED_SECRET	inSymSeed	seed for symmetric key needs *oldParent* private key to recover the seed and generate the symmetric key

Table 52 — TPM2_Rewrap Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PRIVATE	outDuplicate	an object encrypted using symmetric key derived from *outSymSeed*
TPM2B_ENCRYPTED_SECRET	outSymSeed	seed for a symmetric key protected by *newParent* asymmetric key

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

14.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Rewrap_fp.h"
#ifdef TPM_CC_Rewrap  // Conditional expansion of this file
#include "Object_spt_fp.h"
```

Table 53 — TPM2_Rewrap Errors

Error Returns	Meaning
TPM_RC_ATTRIBUTES	*newParent* is not a decryption key
TPM_RC_HANDLE	*oldParent* does not consistent with *inSymSeed*
TPM_RC_INTEGRITY	the integrity check of *inDuplicate* 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 *oldParent*
TPM_RC_TYPE	*oldParent* is not a storage key, or '*newParent* is not a storage key
TPM_RC_VALUE	for an '*oldParent*; 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

```
TPM_RC
TPM2_Rewrap(
    Rewrap_In       *in,            // IN: input parameter list
    Rewrap_Out      *out            // OUT: output parameter list
    )
{
    TPM_RC                  result = TPM_RC_SUCCESS;
    OBJECT                  *oldParent;
    TPM2B_DATA              data;           // symmetric key
    UINT16                  hashSize = 0;
    TPM2B_PRIVATE           privateBlob;    // A temporary private blob
                                            // to transit between old
                                            // and new wrappers

// Input Validation

    if((in->inSymSeed.t.size == 0 && in->oldParent != TPM_RH_NULL)
            || (in->inSymSeed.t.size != 0 && in->oldParent == TPM_RH_NULL))
        return TPM_RC_HANDLE + RC_Rewrap_oldParent;

    if(in->oldParent != TPM_RH_NULL)
    {
        // Get old parent pointer
        oldParent = ObjectGet(in->oldParent);

        // old parent key must be a storage object
        if(!ObjectIsStorage(in->oldParent))
            return TPM_RC_TYPE + RC_Rewrap_oldParent;

        // Decrypt input secret data via asymmetric decryption.  A
        // TPM_RC_VALUE, TPM_RC_KEY or unmarshal errors may be returned at this
        // point
        // See ISO/IEC 11889-1, clause 5.4, "KDF Label Parameters"
        result = CryptSecretDecrypt(in->oldParent, NULL,
                                    "DUPLICATE", &in->inSymSeed, &data);
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
        if(result != TPM_RC_SUCCESS)
            return TPM_RC_VALUE + RC_Rewrap_inSymSeed;

        // Unwrap Outer
        result = UnwrapOuter(in->oldParent, &in->name,
                             oldParent->publicArea.nameAlg, (TPM2B_SEED *) &data,
                             FALSE,
                             in->inDuplicate.t.size, in->inDuplicate.t.buffer);
        if(result != TPM_RC_SUCCESS)
            return RcSafeAddToResult(result, RC_Rewrap_inDuplicate);

        // Copy unwrapped data to temporary variable, remove the integrity field
        hashSize = sizeof(UINT16) +
                   CryptGetHashDigestSize(oldParent->publicArea.nameAlg);
        privateBlob.t.size = in->inDuplicate.t.size - hashSize;
        MemoryCopy(privateBlob.t.buffer, in->inDuplicate.t.buffer + hashSize,
                   privateBlob.t.size, sizeof(privateBlob.t.buffer));
    }
    else
    {
        // No outer wrap from input blob.  Direct copy.
        privateBlob = in->inDuplicate;
    }

    if(in->newParent != TPM_RH_NULL)
    {
        OBJECT          *newParent;
        newParent = ObjectGet(in->newParent);

        // New parent must be a storage object
        if(!ObjectIsStorage(in->newParent))
            return TPM_RC_TYPE + RC_Rewrap_newParent;

        // Make new encrypt key and its associated secret structure.  A
        // TPM_RC_VALUE error may be returned at this point if RSA algorithm is
        // enabled in TPM
        // See ISO/IEC 11889-1, clause 5.4, "KDF Label Parameters"
        out->outSymSeed.t.size = sizeof(out->outSymSeed.t.secret);
        result = CryptSecretEncrypt(in->newParent,
                                    "DUPLICATE", &data, &out->outSymSeed);
        if(result != TPM_RC_SUCCESS) return result;

// Command output
        // Copy temporary variable to output, reserve the space for integrity
        hashSize = sizeof(UINT16) +
                   CryptGetHashDigestSize(newParent->publicArea.nameAlg);
        out->outDuplicate.t.size = privateBlob.t.size;
        MemoryCopy(out->outDuplicate.t.buffer + hashSize, privateBlob.t.buffer,
                   privateBlob.t.size, sizeof(out->outDuplicate.t.buffer));

        // Produce outer wrapper for output
        out->outDuplicate.t.size = ProduceOuterWrap(in->newParent, &in->name,
                                   newParent->publicArea.nameAlg,
                                   (TPM2B_SEED *) &data,
                                   FALSE,
                                   out->outDuplicate.t.size,
                                   out->outDuplicate.t.buffer);

    }
    else  // New parent is a null key so there is no seed
    {
        out->outSymSeed.t.size = 0;

        // Copy privateBlob directly
        out->outDuplicate = privateBlob;
    }
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
106
107         return TPM_RC_SUCCESS;
108     }
109     #endif // CC_Rewrap
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

14.3 TPM2_Import

14.3.1 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).

If encryptedDuplication is SET in objectPublic, then *inSymSeed* and *encryptionKey* shall not be Empty buffers (TPM_RC_ATTRIBUTES). Recovery of the sensitive data of the object occurs in the TPM in a multi--step process in the following order:

a) If *inSymSeed* has a non-zero size:

   1) 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.

      NOTE 1 When recovering the seed from *inSymSeed*, *L* is “DUPLICATE”. (See ISO/IEC 11889-1, clause 5.4, "KDF Label Parameters" for normative KDF label values.)

   2) The integrity value in *duplicate.buffer.integrityOuter* is used to verify the integrity of the inner data blob, which is the remainder of *duplicate.buffer* (TPM_RC_INTEGRITY).

      NOTE 2 The inner data blob will contain a TPMT_SENSITIVE and can contain a TPM2B_DIGEST for the *innerIntegrity*.

   3) The symmetric key recovered in 1) (2)is used to decrypt the inner data blob.

      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.

b) If *encryptionKey* is not an Empty Buffer:

   1) Use *encryptionKey* to decrypt the inner blob.

   2) Use the TPM2B_DIGEST at the start of the inner blob to verify the integrity of the inner blob (TPM_RC_INTEGRITY).

c) Unmarshal the sensitive area

NOTE 4 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 needs to 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()).

Similarly, if the new parent's fixedTPM is set, the *encryptedDuplication* state need only be checked at import.

If the new parent is not fixedTPM, then that object will be loadable on any TPM (including SW versions) on which the new parent exists. This means that, each time an object is loaded under a parent that is not fixedTPM, it is necessary to validate all of the properties of that object. If the parent is fixedTPM, then the new private blob is integrity protected by the TPM that “owns” the parent. So, it is sufficient to validate the object’s properties (attribute and public-private binding) on import and not again.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

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

NOTE 5 The symmetric re-encryption is the normal integrity generation and symmetric encryption applied to a child object.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

14.3.2 Command and Response

Table 54 — TPM2_Import Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 *duplicate* If *symmetricAlg* 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 *duplicate* and the object attributes can be checked.
TPM2B_PRIVATE	duplicate	the symmetrically encrypted duplicate object that may contain an inner symmetric wrapper
TPM2B_ENCRYPTED_SECRET	inSymSeed	symmetric key used to encrypt *duplicate* *inSymSeed* is encrypted/encoded using the algorithms of *newParent*.
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 *encryptionKey* shall be the Empty Buffer.

NOTE Even if the integrity value of the object is not checked on input, the object Name is needed to create the integrity value for the imported object.

Table 55 — TPM2_Import Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PRIVATE	outPrivate	the sensitive area encrypted with the symmetric key of *parentHandle*

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

14.3.3 Detailed Actions

```
1   #include "InternalRoutines.h"
2   #include "Import_fp.h"
3   #ifdef TPM_CC_Import  // Conditional expansion of this file
4   #include "Object_spt_fp.h"
```

Table 56 — TPM2_Import Errors

Error Returns	Meaning
TPM_RC_ASYMMETRIC	non-duplicable storage key represented by *objectPublic* and its parent referenced by *parentHandle* have different public parameters
TPM_RC_ATTRIBUTES	attributes *FixedTPM* and *fixedParent* of *objectPublic* are not both CLEAR; or *inSymSeed* is nonempty and *parentHandle* does not reference a decryption key; or *objectPublic* and *parentHandle* have incompatible or inconsistent attributes; or *encrytpedDuplication* is SET in *objectPublic* but the inner or outer wrapper is missing.
TPM_RC_BINDING	*duplicate* and *objectPublic* are not cryptographically bound
TPM_RC_ECC_POINT	*inSymSeed* is nonempty and ECC point in *inSymSeed* is not on the curve
TPM_RC_HASH	non-duplicable storage key represented by *objectPublic* and its parent referenced by *parentHandle* have different name algorithm
TPM_RC_INSUFFICIENT	*inSymSeed* is nonempty and failed to retrieve ECC point from the secret; or unmarshaling sensitive value from *duplicate* failed the result of *inSymSeed* decryption
TPM_RC_INTEGRITY	*duplicate* integrity is broken
TPM_RC_KDF	*objectPublic* representing decrypting keyed hash object specifies invalid KDF
TPM_RC_KEY	inconsistent parameters of *objectPublic*; or *inSymSeed* is nonempty and *parentHandle* does not reference a key of supported type; or invalid key size in *objectPublic* representing an asymmetric key
TPM_RC_NO_RESULT	*inSymSeed* is nonempty and multiplication resulted in ECC point at infinity
TPM_RC_OBJECT_MEMORY	no available object slot
TPM_RC_SCHEME	inconsistent attributes *decrypt*, *sign*, *restricted* and key's scheme ID in *objectPublic*; or hash algorithm is inconsistent with the scheme ID for keyed hash object
TPM_RC_SIZE	*authPolicy* size does not match digest size of the name algorithm in *objectPublic*; or *symmetricAlg* and *encryptionKey* have different sizes; or *inSymSeed* is nonempty and it size is not consistent with the type of *parentHandle*; or unmarshaling sensitive value from *duplicate* failed
TPM_RC_SYMMETRIC	*objectPublic* 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 *objectPublic*; or non-duplicable storage key represented by *objectPublic* and its parent referenced by *parentHandle* are of different types; or *parentHandle* is not a storage key; or only the public portion of *parentHandle* is loaded; or *objectPublic* and *duplicate* are of different types

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

Error Returns	Meaning
TPM_RC_VALUE	nonempty *inSymSeed* and its numeric value is greater than the modulus of the key referenced by *parentHandle* or *inSymSeed* is larger than the size of the digest produced by the name algorithm of the symmetric key referenced by *parentHandle*
NOTE: Regarding TPM_RC_ATTRIBUTES, if the TPM provides parameter values, the parameter number will indicate *symmetricKey* (missing inner wrapper) or *inSymSeed* (missing outer wrapper).	

```
TPM_RC
TPM2_Import(
    Import_In       *in,            // IN: input parameter list
    Import_Out      *out            // OUT: output parameter list
    )
{

    TPM_RC                  result = TPM_RC_SUCCESS;
    OBJECT                  *parentObject;
    TPM2B_DATA              data;                   // symmetric key
    TPMT_SENSITIVE          sensitive;
    TPM2B_NAME              name;

    UINT16                  innerKeySize = 0;       // encrypt key size for inner
                                                    // wrapper

// Input Validation

    // FixedTPM and fixedParent must be CLEAR
    if(   in->objectPublic.t.publicArea.objectAttributes.fixedTPM == SET
       || in->objectPublic.t.publicArea.objectAttributes.fixedParent == SET)
        return TPM_RC_ATTRIBUTES + RC_Import_objectPublic;

    // Get parent pointer
    parentObject = ObjectGet(in->parentHandle);

    if(!AreAttributesForParent(parentObject))
        return TPM_RC_TYPE + RC_Import_parentHandle;

    if(in->symmetricAlg.algorithm != TPM_ALG_NULL)
    {
        // Get inner wrap key size
        innerKeySize = in->symmetricAlg.keyBits.sym;
        // Input symmetric key must match the size of algorithm.
        if(in->encryptionKey.t.size != (innerKeySize + 7) / 8)
            return TPM_RC_SIZE + RC_Import_encryptionKey;
    }
    else
    {
        // If input symmetric algorithm is NULL, input symmetric key size must
        // be 0 as well
        if(in->encryptionKey.t.size != 0)
            return TPM_RCS_SIZE + RC_Import_encryptionKey;
        // If encryptedDuplication is SET, then the object must have an inner
        // wrapper
        if(in->objectPublic.t.publicArea.objectAttributes.encryptedDuplication)
            return TPM_RCS_ATTRIBUTES + RC_Import_encryptionKey;
    }

    // See if there is an outer wrapper
    if(in->inSymSeed.t.size != 0)
    {
        // Decrypt input secret data via asymmetric decryption. TPM_RC_ATTRIBUTES,
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
58          // TPM_RC_ECC_POINT, TPM_RC_INSUFFICIENT, TPM_RC_KEY, TPM_RC_NO_RESULT,
59          // TPM_RC_SIZE, TPM_RC_VALUE may be returned at this point
60          // See ISO/IEC 11889-1, clause 5.4, "KDF Label Parameters"
61          result = CryptSecretDecrypt(in->parentHandle, NULL, "DUPLICATE",
62                                      &in->inSymSeed, &data);
63          pAssert(result != TPM_RC_BINDING);
64          if(result != TPM_RC_SUCCESS)
65              return RcSafeAddToResult(result, RC_Import_inSymSeed);
66      }
67      else
68      {
69          // If encrytpedDuplication is set, then the object must have an outer
70          // wrapper
71          if(in->objectPublic.t.publicArea.objectAttributes.encryptedDuplication)
72              return TPM_RCS_ATTRIBUTES + RC_Import_inSymSeed;
73          data.t.size = 0;
74      }
75
76      // Compute name of object
77      ObjectComputeName(&(in->objectPublic.t.publicArea), &name);
78
79      // Retrieve sensitive from private.
80      // TPM_RC_INSUFFICIENT, TPM_RC_INTEGRITY, TPM_RC_SIZE may be returned here.
81      result = DuplicateToSensitive(&in->duplicate, &name, in->parentHandle,
82                                    in->objectPublic.t.publicArea.nameAlg,
83                                    (TPM2B_SEED *) &data, &in->symmetricAlg,
84                                    &in->encryptionKey, &sensitive);
85      if(result != TPM_RC_SUCCESS)
86          return RcSafeAddToResult(result, RC_Import_duplicate);
87
88      // If the parent of this object has fixedTPM SET, then fully validate this
89      // object so that validation can be skipped when it is loaded
90      if(parentObject->publicArea.objectAttributes.fixedTPM == SET)
91      {
92          TPM_HANDLE       objectHandle;
93
94          // Perform self check on input public area.  A TPM_RC_SIZE, TPM_RC_SCHEME,
95          // TPM_RC_VALUE, TPM_RC_SYMMETRIC, TPM_RC_TYPE, TPM_RC_HASH,
96          // TPM_RC_ASYMMETRIC, TPM_RC_ATTRIBUTES or TPM_RC_KDF error may be returned
97          // at this point
98          result = PublicAttributesValidation(TRUE, in->parentHandle,
99                                              &in->objectPublic.t.publicArea);
100         if(result != TPM_RC_SUCCESS)
101             return RcSafeAddToResult(result, RC_Import_objectPublic);
102
103         // Create internal object.  A TPM_RC_KEY_SIZE, TPM_RC_KEY or
104         // TPM_RC_OBJECT_MEMORY  error may be returned at this point
105         result = ObjectLoad(TPM_RH_NULL, &in->objectPublic.t.publicArea,
106                             &sensitive, NULL, in->parentHandle, FALSE,
107                             &objectHandle);
108         if(result != TPM_RC_SUCCESS)
109             return result;
110
111         // Don't need the object, just needed the checks to be performed so
112         // flush the object
113         ObjectFlush(objectHandle);
114     }
115
116 // Command output
117
118     // Prepare output private data from sensitive
119     SensitiveToPrivate(&sensitive, &name, in->parentHandle,
120                        in->objectPublic.t.publicArea.nameAlg,
121                        &out->outPrivate);
122
123     return TPM_RC_SUCCESS;
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
124     }
125     #endif // CC_Import
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

15 Asymmetric Primitives

15.1 Introduction

The commands in clause 15 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.

15.2 TPM2_RSA_Encrypt

15.2.1 General Description

This command performs RSA encryption using the indicated padding scheme according to IETF RFC 3447. 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 1 Stiplating 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 can 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 specified in 7.1 of IETF RFC 3447 (PKCS#1). The only supported mask generation is MGF1.
2) TPM_ALG_RSAES – Data is padded as specified in 7.2 of IETF RFC 3447 (PKCS#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 57 — Padding Scheme Selection

keyHandle→scheme	***inScheme***	**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 IETF RFC 3447 (PKCS#1).

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

NOTE 2 *decrypt* needs to 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 3 If it is desired to use a key that had restricted SET, the caller can 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 4 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 won't 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 58 — Message Size Limits Based on Padding

Scheme	Maximum Message Length (*mLen*) in Octets	Comments
TPM_ALG_OAEP	mLen ≤ k – 2hLen – 2	
TPM_ALG_RSAES	mLen ≤ k – 11	
TPM_ALG_NULL	mLen ≤ k	The numeric value of the message must be less than the numeric value of the public modulus (*n*).

NOTES

1) k ≔ the number of byes in the public modulus

2) $hLen$ ≔ 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 5 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 6 Only the public area of *keyHandle* is required to be loaded. A public key can be loaded with any desired scheme. If the scheme is to be changed, a different public area needs to be loaded.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

15.2.2 Command and Response

Table 59 — TPM2_RSA_Encrypt Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit, encrypt, or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
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
TPMT_RSA_DECRYPT+	inScheme	the padding scheme to use if *scheme* associated with *keyHandle* is TPM_ALG_NULL
TPM2B_DATA	label	optional label *L* to be associated with the message Size of the buffer is zero if no label is present

NOTE 1 The *message* 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 *keyHandle*.

NOTE 2 Regarding *label*, see description of label above.

Table 60 — TPM2_RSA_Encrypt Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PUBLIC_KEY_RSA	outData	encrypted output

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

15.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "RSA_Encrypt_fp.h"
#ifdef TPM_CC_RSA_Encrypt  // Conditional expansion of this file
#ifdef TPM_ALG_RSA
```

Table 61 — TPM2_RSA_Encrypt Errors

Error Returns	Meaning
TPM_RC_ATTRIBUTES	*decrypt* attribute is not SET in key referenced by *keyHandle*
TPM_RC_KEY	*keyHandle* 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 *message* is greater than the public modulus of the key referenced by *keyHandle*, or *label* is not a null-terminated string

```
TPM_RC
TPM2_RSA_Encrypt(
    RSA_Encrypt_In      *in,            // IN: input parameter list
    RSA_Encrypt_Out     *out            // OUT: output parameter list
    )
{
    TPM_RC                  result;
    OBJECT                  *rsaKey;
    TPMT_RSA_DECRYPT        *scheme;
    char                    *label = NULL;

// Input Validation

    rsaKey = ObjectGet(in->keyHandle);

    // selected key must be an RSA key
    if(rsaKey->publicArea.type != TPM_ALG_RSA)
        return TPM_RC_KEY + RC_RSA_Encrypt_keyHandle;

    // selected key must have the decryption attribute
    if(rsaKey->publicArea.objectAttributes.decrypt != SET)
        return TPM_RC_ATTRIBUTES + RC_RSA_Encrypt_keyHandle;

    // Is there a label?
    if(in->label.t.size > 0)
    {
        // label is present, so make sure that is it NULL-terminated
        if(in->label.t.buffer[in->label.t.size - 1] != 0)
            return TPM_RC_VALUE + RC_RSA_Encrypt_label;
        label = (char *)in->label.t.buffer;
    }

// Command Output

    // Select a scheme for encryption
    scheme = CryptSelectRSAScheme(in->keyHandle, &in->inScheme);
    if(scheme == NULL)
        return TPM_RC_SCHEME + RC_RSA_Encrypt_inScheme;

    // Encryption.  TPM_RC_VALUE, or TPM_RC_SCHEME errors my be returned buy
    // CryptEncyptRSA. Note: It can also return TPM_RC_ATTRIBUTES if the key does
    // not have the decrypt attribute but that was checked above.
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
47          out->outData.t.size = sizeof(out->outData.t.buffer);
48          result = CryptEncryptRSA(&out->outData.t.size, out->outData.t.buffer, rsaKey,
49                                   scheme, in->message.t.size, in->message.t.buffer,
50                                   label);
51          return result;
52      }
53      #endif
54      #endif // CC_RSA_Encrypt
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

15.3 TPM2_RSA_Decrypt

15.3.1 General Description

This command performs RSA decryption using the indicated padding scheme according to IETF RFC 3447 ((PKCS#1).

The scheme selection for this command is the same as for TPM2_RSA_Encrypt() and is shown in Table 57.

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 specified in IETF RFC 3447 (PKCS#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 during encryption, the *label* parameter is required to be present in the decryption process and *label* is required to be the same in both cases. If label is not the same, the decrypt operation is very likely to fail ((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 *inScheme* is used, and the scheme requires a hash algorithm it may not be TPM_ALG_NULL.

If the 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

15.3.2 Command and Response

Table 62 — TPM2_RSA_Decrypt Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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
TPMT_RSA_DECRYPT+	inScheme	the padding scheme to use if *scheme* associated with *keyHandle* is TPM_ALG_NULL
TPM2B_DATA	label	label whose association with the message is to be verified
NOTE Regarding *cipherText*, an encrypted RSA data block is the size of the public modulus.		

Table 63 — TPM2_RSA_Decrypt Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_PUBLIC_KEY_RSA	message	decrypted output

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

15.3.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "RSA_Decrypt_fp.h"
#ifdef TPM_CC_RSA_Decrypt  // Conditional expansion of this file
#ifdef TPM_ALG_RSA
```

Table 64 — TPM2_RSA_Decrypt Errors

Error Returns	Meaning
TPM_RC_BINDING	The public an private parts of the key are not properly bound
TPM_RC_KEY	*keyHandle* does not reference an unrestricted decrypt key
TPM_RC_SCHEME	incorrect input scheme, or the chosen *scheme* is not a valid RSA decrypt scheme
TPM_RC_SIZE	*cipherText* is not the size of the modulus of key referenced by *keyHandle*
TPM_RC_VALUE	*label* is not a null terminated string or the value of *cipherText* is greater that the modulus of *keyHandle*

```
TPM_RC
TPM2_RSA_Decrypt(
    RSA_Decrypt_In      *in,            // IN: input parameter list
    RSA_Decrypt_Out     *out            // OUT: output parameter list
    )
{
    TPM_RC                      result;
    OBJECT                     *rsaKey;
    TPMT_RSA_DECRYPT           *scheme;
    char                       *label = NULL;

// Input Validation

    rsaKey = ObjectGet(in->keyHandle);

    // The selected key must be an RSA key
    if(rsaKey->publicArea.type != TPM_ALG_RSA)
        return TPM_RC_KEY + RC_RSA_Decrypt_keyHandle;

    // The selected key must be an unrestricted decryption key
    if(   rsaKey->publicArea.objectAttributes.restricted == SET
       || rsaKey->publicArea.objectAttributes.decrypt == CLEAR)
        return TPM_RC_ATTRIBUTES + RC_RSA_Decrypt_keyHandle;

    // NOTE: Proper operation of this command requires that the sensitive area
    // of the key is loaded. This is assured because authorization is required
    // to use the sensitive area of the key. In order to check the authorization,
    // the sensitive area has to be loaded, even if authorization is with policy.

    // If label is present, make sure that it is a NULL-terminated string
    if(in->label.t.size > 0)
    {
        // Present, so make sure that it is NULL-terminated
        if(in->label.t.buffer[in->label.t.size - 1] != 0)
            return TPM_RC_VALUE + RC_RSA_Decrypt_label;
        label = (char *)in->label.t.buffer;
    }

// Command Output

```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
45      // Select a scheme for decrypt.
46      scheme = CryptSelectRSAScheme(in->keyHandle, &in->inScheme);
47      if(scheme == NULL)
48          return TPM_RC_SCHEME + RC_RSA_Decrypt_inScheme;
49
50      // Decryption.  TPM_RC_VALUE, TPM_RC_SIZE, and TPM_RC_KEY error may be
51      // returned by CryptDecryptRSA.
52      // NOTE: CryptDecryptRSA can also return TPM_RC_ATTRIBUTES or TPM_RC_BINDING
53      // when the key is not a decryption key but that was checked above.
54      out->message.t.size = sizeof(out->message.t.buffer);
55      result = CryptDecryptRSA(&out->message.t.size, out->message.t.buffer, rsaKey,
56                               scheme, in->cipherText.t.size,
57                               in->cipherText.t.buffer,
58                               label);
59
60      return result;
61  }
62  #endif
63  #endif // CC_RSA_Decrypt
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

15.4 TPM2_ECDH_KeyGen

15.4.1 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 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

15.4.2 Command and Response

Table 65 — TPM2_ECDH_KeyGen Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or encrypt session is present; otherwise, TPM_ST_NO_SESSIONS
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 66 — TPM2_ECDH_KeyGen Response

Type	Name	Description
TPM_ST	tag	see clause 7
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)

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

15.4.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "ECDH_KeyGen_fp.h"
#ifdef TPM_CC_ECDH_KeyGen  // Conditional expansion of this file
#ifdef TPM_ALG_ECC
```

Table 67 — TPM2_ECDH_KeyGen Errors

Error Returns	Meaning
TPM_RC_KEY	*keyHandle* does not reference a non-restricted decryption ECC key

```
TPM_RC
TPM2_ECDH_KeyGen(
    ECDH_KeyGen_In      *in,            // IN: input parameter list
    ECDH_KeyGen_Out     *out            // OUT: output parameter list
    )
{
    OBJECT                  *eccKey;
    TPM2B_ECC_PARAMETER      sensitive;
    TPM_RC                   result;

// Input Validation

    eccKey = ObjectGet(in->keyHandle);

    // Input key must be a non-restricted, decrypt ECC key
    if(   eccKey->publicArea.type != TPM_ALG_ECC)
        return TPM_RCS_KEY + RC_ECDH_KeyGen_keyHandle;

    if(     eccKey->publicArea.objectAttributes.restricted == SET
       ||   eccKey->publicArea.objectAttributes.decrypt != SET
      )
        return TPM_RC_KEY + RC_ECDH_KeyGen_keyHandle;

// Command Output
    do
    {
        // Create ephemeral ECC key
        CryptNewEccKey(eccKey->publicArea.parameters.eccDetail.curveID,
                       &out->pubPoint.t.point, &sensitive);

        out->pubPoint.t.size = TPMS_ECC_POINT_Marshal(&out->pubPoint.t.point,
                               NULL, NULL);

        // Compute Z
        result = CryptEccPointMultiply(&out->zPoint.t.point,
                                       eccKey->publicArea.parameters.eccDetail.curveID,
                                       &sensitive, &eccKey->publicArea.unique.ecc);
        // The point in the key is not on the curve. Indicate that the key is bad.
        if(result == TPM_RC_ECC_POINT)
            return TPM_RC_KEY + RC_ECDH_KeyGen_keyHandle;
        // The other possible error is TPM_RC_NO_RESULT indicating that the
        // multiplication resulted in the point at infinity, so get a new
        // random key and start over (hardly ever happens).
    }
    while(result == TPM_RC_NO_RESULT);

    if(result == TPM_RC_SUCCESS)
        // Marshal the values to generate the point.
        out->zPoint.t.size = TPMS_ECC_POINT_Marshal(&out->zPoint.t.point,
                                                    NULL, NULL);
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
55
56          return result;
57      }
58      #endif
59      #endif // CC_ECDH_KeyGen
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

15.5 TPM2_ECDH_ZGen

15.5.1 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

15.5.2 Command and Response

Table 68 — TPM2_ECDH_ZGen Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 69 — TPM2_ECDH_ZGen Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	outPoint	X and Y coordinates of the product of the multiplication $Z = (x_Z, y_Z) := [hd_S]Q_B$

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

15.5.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "ECDH_ZGen_fp.h"
#ifdef TPM_CC_ECDH_ZGen  // Conditional expansion of this file
#ifdef TPM_ALG_ECC
```

Table 70 — TPM2_ECDH_ZGen Errors

Error Returns	Meaning
TPM_RC_ATTRIBUTES	key referenced by *keyA* is restricted or not a decrypt key
TPM_RC_KEY	key referenced by *keyA* is not an ECC key
TPM_RC_NO_RESULT	multiplying *inPoint* resulted in a point at infinity
TPM_RC_SCHEME	the scheme of the key referenced by *keyA* is not TPM_ALG_NULL, TPM_ALG_ECDH,

```
TPM_RC
TPM2_ECDH_ZGen(
    ECDH_ZGen_In    *in,            // IN: input parameter list
    ECDH_ZGen_Out   *out            // OUT: output parameter list
    )
{
    TPM_RC                  result;
    OBJECT                 *eccKey;

// Input Validation

    eccKey = ObjectGet(in->keyHandle);

    // Input key must be a non-restricted, decrypt ECC key
    if(   eccKey->publicArea.type != TPM_ALG_ECC)
        return TPM_RCS_KEY + RC_ECDH_ZGen_keyHandle;

    if(     eccKey->publicArea.objectAttributes.restricted == SET
       ||   eccKey->publicArea.objectAttributes.decrypt != SET
      )
        return TPM_RC_KEY + RC_ECDH_ZGen_keyHandle;

    // Make sure the scheme allows this use
    if(     eccKey->publicArea.parameters.eccDetail.scheme.scheme != TPM_ALG_ECDH
        &&  eccKey->publicArea.parameters.eccDetail.scheme.scheme != TPM_ALG_NULL)
        return TPM_RC_SCHEME + RC_ECDH_ZGen_keyHandle;

// Command Output

    // Compute Z. TPM_RC_ECC_POINT or TPM_RC_NO_RESULT may be returned here.
    result = CryptEccPointMultiply(&out->outPoint.t.point,
                                   eccKey->publicArea.parameters.eccDetail.curveID,
                                   &eccKey->sensitive.sensitive.ecc,
                                   &in->inPoint.t.point);
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, RC_ECDH_ZGen_inPoint);

    out->outPoint.t.size = TPMS_ECC_POINT_Marshal(&out->outPoint.t.point,
                                                  NULL, NULL);

    return TPM_RC_SUCCESS;
}
#endif
#endif // CC_ECDH_ZGen
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

15.6 TPM2_ECC_Parameters

15.6.1 General Description

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

15.6.2 Command and Response

Table 71 — TPM2_ECC_Parameters Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_ECC_Parameters
TPMI_ECC_CURVE	curveID	parameter set selector

Table 72 — TPM2_ECC_Parameters Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPMS_ALGORITHM_DETAIL_ECC	parameters	ECC parameters for the selected curve

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

15.6.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "ECC_Parameters_fp.h"
#ifdef TPM_CC_ECC_Parameters  // Conditional expansion of this file
#ifdef TPM_ALG_ECC
```

Table 73 — TPM2_ECC_Parameters Errors

Error Returns	Meaning
TPM_RC_VALUE	Unsupported ECC curve ID

```
TPM_RC
TPM2_ECC_Parameters(
    ECC_Parameters_In   *in,            // IN: input parameter list
    ECC_Parameters_Out  *out            // OUT: output parameter list
    )
{
// Command Output

    // Get ECC curve parameters
    if(CryptEccGetParameters(in->curveID, &out->parameters))
        return TPM_RC_SUCCESS;
    else
        return TPM_RC_VALUE + RC_ECC_Parameters_curveID;
}
#endif
#endif // CC_ECC_Parameters
```

15.7 TPM2_ZGen_2Phase

15.7.1 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 in the TCG Algorithm Registry, Revision 1.15, 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 needs to support the key exchange protocol of SM2, Part 3.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

For TPM_ALG_ECDH *outZ1* will be Z_s and *outZ2* will Z_e as defined in 6.1.1.2 of SP800-56A.

NOTE 2 An unrestricted decryption key using ECDH can 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 4 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

15.7.2 Command and Response

Table 74 — TPM2_ZGen_2Phase Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 75 — 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)

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

15.7.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "ZGen_2Phase_fp.h"
#ifdef TPM_CC_ZGen_2Phase  // Conditional expansion of this file
```

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

Table 76 — TPM2_ZGen_2Phase Errors

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

```
TPM_RC
TPM2_ZGen_2Phase(
    ZGen_2Phase_In      *in,            // IN: input parameter list
    ZGen_2Phase_Out     *out            // OUT: output parameter list
    )
{
    TPM_RC                   result;
    OBJECT                  *eccKey;
    TPM2B_ECC_PARAMETER      r;
    TPM_ALG_ID               scheme;

// Input Validation

    eccKey = ObjectGet(in->keyA);

    // keyA must be an ECC key
    if(eccKey->publicArea.type != TPM_ALG_ECC)
        return TPM_RC_KEY + RC_ZGen_2Phase_keyA;

    // keyA must not be restricted and must be a decrypt key
    if(   eccKey->publicArea.objectAttributes.restricted == SET
       || eccKey->publicArea.objectAttributes.decrypt != SET
      )
        return TPM_RC_ATTRIBUTES + RC_ZGen_2Phase_keyA;

    // if the scheme of keyA is TPM_ALG_NULL, then use the input scheme; otherwise
    // the input scheme must be the same as the scheme of keyA
    scheme = eccKey->publicArea.parameters.asymDetail.scheme.scheme;
    if(scheme != TPM_ALG_NULL)
    {
        if(scheme != in->inScheme)
            return TPM_RC_SCHEME + RC_ZGen_2Phase_inScheme;
    }
    else
        scheme = in->inScheme;
    if(scheme == TPM_ALG_NULL)
        return TPM_RC_SCHEME + RC_ZGen_2Phase_inScheme;

    // Input points must be on the curve of keyA
    if(!CryptEccIsPointOnCurve(eccKey->publicArea.parameters.eccDetail.curveID,
                               &in->inQsB.t.point))
        return TPM_RC_ECC_POINT + RC_ZGen_2Phase_inQsB;

```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
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_SCHEME)
67          return TPM_RC_SCHEME + RC_ZGen_2Phase_inScheme;
68
69      if(result == TPM_RC_SUCCESS)
70          CryptEndCommit(in->counter);
71
72      return result;
73  }
74  #endif
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

16 Symmetric Primitives

16.1 Introduction

The commands in clause 16 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:

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

Table 77 — Symmetric Chaining Process

Mode	Chaining process
TPM_ALG_CTR	The TPM will increment the entire IV provided by the caller. The next count value will be returned to the caller as *ivOut*. This can be the input value to the next encrypt or decrypt operation. *ivIn* is required to be the size of a block encrypted by the selected algorithm and key combination. If the size of *ivIn* is not correct, the TPM shall return TPM_RC_SIZE. *ivOut* will be the size of a cipher block and not the size of the last encrypted block. All the bits of the IV are incremented as if it were an unsigned integer.
TPM_ALG_OFB	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. *ivOut* will be the value that was XORed with the last plaintext block. That value can be used as the *ivIn* for a next buffer. *ivIn* is required to be the size of a block encrypted by the selected algorithm and key combination. If the size of *ivIn* is not correct, the TPM shall return TPM_RC_SIZE. *ivOut* will be the size of a cipher block and not the size of the last encrypted block.
TPM_ALG_CBC	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. Even though the last ciphertext block is evident in the encrypted data, it is also returned in *ivOut*. *ivIn* is required to be the size of a block encrypted by the selected algorithm and key combination. If the size of *ivIn* is not correct, the TPM shall return TPM_RC_SIZE. *inData* is required to be an even multiple of the block encrypted by the selected algorithm and key combination. If the size of *inData* is not correct, the TPM shall return TPM_RC_SIZE.
TPM_ALG_CFB	Similar to CBC in that the last ciphertext block is an input to the encryption of the next block. *ivOut* will be the value that was XORed with the last plaintext block. That value can be used as the *ivIn* for a next buffer. *ivIn* is required to be the size of a block encrypted by the selected algorithm and key combination. If the size of *ivIn* is not correct, the TPM shall return TPM_RC_SIZE. *ivOut* will be the size of a cipher block and not the size of the last encrypted block.
TPM_ALG_ECB	Electronic Codebook (ECB) has no chaining. Each block of plaintext is encrypted using the key. ECB does not support chaining and *ivIn* shall be the Empty Buffer. *ivOut* will be the Empty Buffer. *inData* is required to be an even multiple of the block encrypted by the selected algorithm and key combination. If the size of *inData* is not correct, the TPM shall return TPM_RC_SIZE.

EXAMPLE 1 Regarding TPM_ALG_CTR, AES stipulates that *ivIn* be 128 bits (16 octets).

NOTE Regarding TPM_ALG_CTR, *ivOut* will be the value of the counter after the last block is encrypted.

EXAMPLE 2 Regarding TPM_ALG_CTR, if *ivIn* were $00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00_{16}$ and four data blocks were encrypted, *ivOut* will have a value of $00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 00\ 04_{16}$.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

16.2 TPM2_EncryptDecrypt

16.2.1 General Description

This command performs symmetric encryption or decryption.

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 different, *mode* overrides the mode of the key.

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_CANCELED. In such case, *outData* may be less than *inData*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

16.2.2 Command and Response

Table 78 — TPM2_EncryptDecrypt Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 79 — TPM2_EncryptDecrypt Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_MAX_BUFFER	outData	encrypted or decrypted output
TPM2B_IV	ivOut	chaining value to use for IV in next round

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

16.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "EncryptDecrypt_fp.h"
#ifdef TPM_CC_EncryptDecrypt  // Conditional expansion of this file
```

Table 80 — TPM2_EncryptDecrypt Errors

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

```
TPM_RC
TPM2_EncryptDecrypt(
    EncryptDecrypt_In   *in,            // IN: input parameter list
    EncryptDecrypt_Out  *out            // OUT: output parameter list
    )
{
    OBJECT              *symKey;
    UINT16              keySize;
    UINT16              blockSize;
    BYTE                *key;
    TPM_ALG_ID          alg;

// Input Validation
    symKey = ObjectGet(in->keyHandle);

    // The input key should be a symmetric decrypt key.
    if(    symKey->publicArea.type != TPM_ALG_SYMCIPHER
       || symKey->attributes.publicOnly == SET)
        return TPM_RC_KEY + RC_EncryptDecrypt_keyHandle;

    // If the input mode is TPM_ALG_NULL, use the key's mode
    if( in->mode == TPM_ALG_NULL)
        in->mode = symKey->publicArea.parameters.symDetail.sym.mode.sym;

    // If the key is restricted, the input symmetric mode should match the key's
    // symmetric mode.
    if(   symKey->publicArea.objectAttributes.restricted == SET
       && symKey->publicArea.parameters.symDetail.sym.mode.sym != in->mode)
        return TPM_RC_VALUE + RC_EncryptDecrypt_mode;

    // If the mode is null, then we have a problem.
    // Note: Construction of a TPMT_SYM_DEF does not allow the 'mode' to be
    // TPM_ALG_NULL so setting in->mode to the mode of the key should have
    // produced a valid mode. However, this is suspenders.
    if(in->mode == TPM_ALG_NULL)
        return TPM_RC_VALUE + RC_EncryptDecrypt_mode;

    // The input iv for ECB mode should be null.  All the other modes should
    // have an iv size same as encryption block size

    keySize = symKey->publicArea.parameters.symDetail.sym.keyBits.sym;
    alg = symKey->publicArea.parameters.symDetail.sym.algorithm;
    blockSize = CryptGetSymmetricBlockSize(alg, keySize);
    if(   (in->mode == TPM_ALG_ECB && in->ivIn.t.size != 0)
       || (in->mode != TPM_ALG_ECB && in->ivIn.t.size != blockSize))
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
49          return TPM_RC_SIZE + RC_EncryptDecrypt_ivIn;
50
51      // The input data size of CBC mode or ECB mode must be an even multiple of
52      // the symmetric algorithm's block size
53      if(   (in->mode == TPM_ALG_CBC || in->mode == TPM_ALG_ECB)
54         && (in->inData.t.size % blockSize) != 0)
55          return TPM_RC_SIZE + RC_EncryptDecrypt_inData;
56
57      // Copy IV
58      // Note: This is copied here so that the calls to the encrypt/decrypt functions
59      // will modify the output buffer, not the input buffer
60      out->ivOut = in->ivIn;
61
62  // Command Output
63
64      key = symKey->sensitive.sensitive.sym.t.buffer;
65      // For symmetric encryption, the cipher data size is the same as plain data
66      // size.
67      out->outData.t.size = in->inData.t.size;
68      if(in->decrypt == YES)
69      {
70          // Decrypt data to output
71          CryptSymmetricDecrypt(out->outData.t.buffer,
72                                alg,
73                                keySize, in->mode, key,
74                                &(out->ivOut),
75                                in->inData.t.size,
76                                in->inData.t.buffer);
77      }
78      else
79      {
80          // Encrypt data to output
81          CryptSymmetricEncrypt(out->outData.t.buffer,
82                                alg,
83                                keySize,
84                                in->mode, key,
85                                &(out->ivOut),
86                                in->inData.t.size,
87                                in->inData.t.buffer);
88      }
89
90      return TPM_RC_SUCCESS;
91  }
92  #endif // CC_EncryptDecrypt
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

16.3 TPM2_Hash

16.3.1 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

16.3.2 Command and Response

Table 81 — TPM2_Hash Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit, decrypt, or encrypt session is present; otherwise, TPM_ST_NO_SESSIONS
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 82 — TPM2_Hash Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DIGEST	outHash	results
TPMT_TK_HASHCHECK	validation	ticket indicating that the sequence of octets used to compute *outDigest* did not start with TPM_GENERATED_VALUE will be a NULL ticket if the digest may not be signed with a restricted key

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

16.3.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Hash_fp.h"
#ifdef TPM_CC_Hash  // Conditional expansion of this file
TPM_RC
TPM2_Hash(
    Hash_In         *in,            // IN: input parameter list
    Hash_Out        *out            // OUT: output parameter list
    )
{
    HASH_STATE          hashState;

// Command Output

    // Output hash
        // Start hash stack
    out->outHash.t.size = CryptStartHash(in->hashAlg, &hashState);
        // Adding hash data
    CryptUpdateDigest2B(&hashState, &in->data.b);
        // Complete hash
    CryptCompleteHash2B(&hashState, &out->outHash.b);

    // Output ticket
    out->validation.tag = TPM_ST_HASHCHECK;
    out->validation.hierarchy = in->hierarchy;

    if(in->hierarchy == TPM_RH_NULL)
    {
        // Ticket is not required
        out->validation.hierarchy = TPM_RH_NULL;
        out->validation.digest.t.size = 0;
    }
    else if(  in->data.t.size >= sizeof(TPM_GENERATED)
            && !TicketIsSafe(&in->data.b))
    {
        // Ticket is not safe
        out->validation.hierarchy = TPM_RH_NULL;
        out->validation.digest.t.size = 0;
    }
    else
    {
        // Compute ticket
        TicketComputeHashCheck(in->hierarchy, in->hashAlg,
                               &out->outHash, &out->validation);
    }

    return TPM_RC_SUCCESS;
}
#endif // CC_Hash
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

16.4 TPM2_HMAC

16.4.1 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 can only have one of sign or decrypt SET and the default scheme cannot 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 2 A key can 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 needs to be specified.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

16.4.2 Command and Response

Table 83 — TPM2_HMAC Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 84 — TPM2_HMAC Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DIGEST	outHMAC	the returned HMAC in a sized buffer

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

16.4.3 Detailed Actions

```
1   #include "InternalRoutines.h"
2   #include "HMAC_fp.h"
3   #ifdef TPM_CC_HMAC  // Conditional expansion of this file
```

Table 85 — TPM2_HMAC Errors

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

```
4   TPM_RC
5   TPM2_HMAC(
6       HMAC_In         *in,            // IN: input parameter list
7       HMAC_Out        *out            // OUT: output parameter list
8       )
9   {
10      HMAC_STATE              hmacState;
11      OBJECT                 *hmacObject;
12      TPMI_ALG_HASH           hashAlg;
13      TPMT_PUBLIC            *publicArea;
14
15  // Input Validation
16
17      // Get HMAC key object and public area pointers
18      hmacObject = ObjectGet(in->handle);
19      publicArea = &hmacObject->publicArea;
20
21      // Make sure that the key is an HMAC signing key
22      if(publicArea->type != TPM_ALG_KEYEDHASH)
23          return TPM_RC_TYPE + RC_HMAC_handle;
24      if(publicArea->objectAttributes.sign != SET)
25          return TPM_RC_ATTRIBUTES + RC_HMAC_handle;
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
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
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      }
62      #endif // CC_HMAC
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

17 Random Number Generator

17.1 TPM2_GetRandom

17.1.1 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, as long as the value of *bytesRequested* is not greater than the maximum digest size, 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 can 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

17.1.2 Command and Response

Table 86 — TPM2_GetRandom Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or encrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_GetRandom
UINT16	bytesRequested	number of octets to return

Table 87 — TPM2_GetRandom Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DIGEST	randomBytes	the random octets

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

17.1.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "GetRandom_fp.h"
#ifdef TPM_CC_GetRandom  // Conditional expansion of this file
TPM_RC
TPM2_GetRandom(
    GetRandom_In    *in,            // IN: input parameter list
    GetRandom_Out   *out            // OUT: output parameter list
    )
{
// Command Output

    // if the requested bytes exceed the output buffer size, generates the
    // maximum bytes that the output buffer allows
    if(in->bytesRequested > sizeof(TPMU_HA))
        out->randomBytes.t.size = sizeof(TPMU_HA);
    else
        out->randomBytes.t.size = in->bytesRequested;

    CryptGenerateRandom(out->randomBytes.t.size, out->randomBytes.t.buffer);

    return TPM_RC_SUCCESS;
}
#endif // CC_GetRandom
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

17.2 TPM2_StirRandom

17.2.1 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

17.2.2 Command and Response

Table 88 — TPM2_StirRandom Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_StirRandom {NV}
TPM2B_SENSITIVE_DATA	inData	additional information

Table 89 — TPM2_StirRandom Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

17.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "StirRandom_fp.h"
#ifdef TPM_CC_StirRandom  // Conditional expansion of this file
TPM_RC
TPM2_StirRandom(
    StirRandom_In   *in             // IN: input parameter list
    )
{
// Internal Data Update
    CryptStirRandom(in->inData.t.size, in->inData.t.buffer);

    return TPM_RC_SUCCESS;
}
#endif // CC_StirRandom
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

18 Hash/HMAC/Event Sequences

18.1 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 ISO/IEC 11889-1, clause 32.4, “Hash, HMAC, and Event Sequences”.

18.2 TPM2_HMAC_Start

18.2.1 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 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 90 — Hash Selection Matrix

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

NOTE 1 The scheme for the handle can only be TPM_ALG_NULL if both sign and decrypt are SET.

NOTE 2 A hash algorithm is needed for the HMAC.

NOTE 3 A restricted key needs to have a scheme with a valid hash algorithm. A restricted key cannot have both *sign* and *decrypt* SET

NOTE 4 The scheme for a restricted key cannot be overridden.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

18.2.2 Command and Response

Table 91 — TPM2_HMAC_Start Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 92 — TPM2_HMAC_Start Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPMI_DH_OBJECT	sequenceHandle	a handle to reference the sequence

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

18.2.3 Detailed Actions

```
1   #include "InternalRoutines.h"
2   #include "HMAC_Start_fp.h"
3   #ifdef TPM_CC_HMAC_Start  // Conditional expansion of this file
```

Table 93 — TPM2_HMAC_Start Errors

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

```
4   TPM_RC
5   TPM2_HMAC_Start(
6       HMAC_Start_In   *in,            // IN: input parameter list
7       HMAC_Start_Out  *out            // OUT: output parameter list
8       )
9   {
10      OBJECT                  *hmacObject;
11      TPMT_PUBLIC             *publicArea;
12      TPM_ALG_ID               hashAlg;
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_Start_handle;
23      if(publicArea->objectAttributes.sign != SET)
24          return TPM_RC_ATTRIBUTES + RC_HMAC_Start_handle;
25
26      // Assume that the key default scheme is used
27      hashAlg = publicArea->parameters.keyedHashDetail.scheme.details.hmac.hashAlg;
28
29      // if the key is restricted, then need to use the scheme of the key and the
30      // input algorithm must be TPM_ALG_NULL or the same as the key scheme
31      if(publicArea->objectAttributes.restricted == SET)
32      {
33          if(in->hashAlg != TPM_ALG_NULL && in->hashAlg != hashAlg)
34              hashAlg = TPM_ALG_NULL;
35      }
36      else
37      {
38          // for a non-restricted key, use hashAlg if it is provided;
39          if(in->hashAlg != TPM_ALG_NULL)
40              hashAlg = in->hashAlg;
41      }
42      // if the algorithm selection ended up with TPM_ALG_NULL, then either the
43      // schemes are not compatible or no hash was provided and both conditions
44      // are errors.
45      if(hashAlg == TPM_ALG_NULL)
46          return TPM_RC_VALUE + RC_HMAC_Start_hashAlg;
47
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
48  // Internal Data Update
49
50      // Create a HMAC sequence object. A TPM_RC_OBJECT_MEMORY error may be
51      // returned at this point
52      return ObjectCreateHMACSequence(hashAlg,
53                                      in->handle,
54                                      &in->auth,
55                                      &out->sequenceHandle);
56  }
57  #endif // CC_HMAC_Start
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

18.3 TPM2_HashSequenceStart

18.3.1 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 context or an Event Sequence context. Additionally, it will assign a handle to the context and set the *authValue* of the context to the value in *auth*. A sequence context for an Event (*hashAlg* = TPM_ALG_NULL) contains a hash context for each of the PCR banks implemented on the TPM.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

18.3.2 Command and Response

Table 94 — TPM2_HashSequenceStart Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
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 95 — TPM2_HashSequenceStart Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPMI_DH_OBJECT	sequenceHandle	a handle to reference the sequence

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

18.3.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "HashSequenceStart_fp.h"
#ifdef TPM_CC_HashSequenceStart  // Conditional expansion of this file
```

Table 96 — TPM2_HashSequenceStart Errors

Error Returns	Meaning
TPM_RC_OBJECT_MEMORY	no space to create an internal object

```
TPM_RC
TPM2_HashSequenceStart(
    HashSequenceStart_In    *in,            // IN: input parameter list
    HashSequenceStart_Out   *out            // OUT: output parameter list
    )
{
// Internal Data Update

    if(in->hashAlg == TPM_ALG_NULL)
        // Start a event sequence.  A TPM_RC_OBJECT_MEMORY error may be
        // returned at this point
        return ObjectCreateEventSequence(&in->auth, &out->sequenceHandle);

    // Start a hash sequence.  A TPM_RC_OBJECT_MEMORY error may be
    // returned at this point
    return ObjectCreateHashSequence(in->hashAlg, &in->auth, &out->sequenceHandle);
}
#endif // CC_HashSequenceStart
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

18.4 TPM2_SequenceUpdate

18.4.1 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 1 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 2 This requirement allows the TPM to validate that the first block is safe to sign without having to accumulate octets over multiple calls.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

18.4.2 Command and Response

Table 97 — TPM2_SequenceUpdate Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 98 — TPM2_SequenceUpdate Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

18.4.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "SequenceUpdate_fp.h"
#ifdef TPM_CC_SequenceUpdate  // Conditional expansion of this file
```

Table 99 — TPM2_SequenceUpdate Errors

Error Returns	Meaning
TPM_RC_MODE	*sequenceHandle* does not reference a hash or HMAC sequence object

```
TPM_RC
TPM2_SequenceUpdate(
    SequenceUpdate_In   *in             // IN: input parameter list
    )
{
    OBJECT                  *object;

// Input Validation

    // Get sequence object pointer
    object = ObjectGet(in->sequenceHandle);

    // Check that referenced object is a sequence object.
    if(!ObjectIsSequence(object))
        return TPM_RC_MODE + RC_SequenceUpdate_sequenceHandle;

// Internal Data Update

    if(object->attributes.eventSeq == SET)
    {
        // Update event sequence object
        UINT32           i;
        HASH_OBJECT     *hashObject = (HASH_OBJECT *)object;
        for(i = 0; i < HASH_COUNT; i++)
        {
            // Update sequence object
            CryptUpdateDigest2B(&hashObject->state.hashState[i], &in->buffer.b);
        }
    }
    else
    {
        HASH_OBJECT    *hashObject = (HASH_OBJECT *)object;

        // Update hash/HMAC sequence object
        if(hashObject->attributes.hashSeq == SET)
        {
            // Is this the first block of the sequence
            if(hashObject->attributes.firstBlock == CLEAR)
            {
                // If so, indicate that first block was received
                hashObject->attributes.firstBlock = SET;

                // Check the first block to see if the first block can contain
                // the TPM_GENERATED_VALUE.  If it does, it is not safe for
                // a ticket.
                if(TicketIsSafe(&in->buffer.b))
                    hashObject->attributes.ticketSafe = SET;
            }
            // Update sequence object hash/HMAC stack
            CryptUpdateDigest2B(&hashObject->state.hashState[0], &in->buffer.b);
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
54
55          }
56          else if(object->attributes.hmacSeq == SET)
57          {
58              HASH_OBJECT     *hashObject = (HASH_OBJECT *)object;
59
60              // Update sequence object hash/HMAC stack
61              CryptUpdateDigest2B(&hashObject->state.hmacState, &in->buffer.b);
62          }
63      }
64
65      return TPM_RC_SUCCESS;
66  }
67  #endif // CC_SequenceUpdate
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

18.5 TPM2_SequenceComplete

18.5.1 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.

For a hash sequence, 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 *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.

NOTE 3 The ticket is only needed for a signing operation that uses a restricted signing key. It is always returned, but can be ignored if not needed.

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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

18.5.2 Command and Response

Table 100 — TPM2_SequenceComplete Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 101 — TPM2_SequenceComplete Response

Type	Name	Description
TPM_ST	tag	see clause 7
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 *outDigest* did not start with TPM_GENERATED_VALUE This is a NULL Ticket when the sequence is HMAC.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

18.5.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "SequenceComplete_fp.h"
#ifdef TPM_CC_SequenceComplete  // Conditional expansion of this file
#include <Platform.h>
```

Table 102 — TPM2_SequenceComplete Errors

Error Returns	Meaning
TPM_RC_TYPE	*sequenceHandle* does not reference a hash or HMAC sequence object

```
TPM_RC
TPM2_SequenceComplete(
    SequenceComplete_In     *in,            // IN: input parameter list
    SequenceComplete_Out    *out            // OUT: output parameter list
    )
{
    OBJECT                      *object;

// Input validation

    // Get hash object pointer
    object = ObjectGet(in->sequenceHandle);

    // input handle must be a hash or HMAC sequence object.
    if(   object->attributes.hashSeq == CLEAR
       && object->attributes.hmacSeq == CLEAR)
        return TPM_RC_MODE + RC_SequenceComplete_sequenceHandle;

// Command Output

    if(object->attributes.hashSeq == SET)          // sequence object for hash
    {
        // Update last piece of data
        HASH_OBJECT     *hashObject = (HASH_OBJECT *)object;

       // Get the hash algorithm before the algorithm is lost in CryptCompleteHash
        TPM_ALG_ID       hashAlg = hashObject->state.hashState[0].state.hashAlg;

        CryptUpdateDigest2B(&hashObject->state.hashState[0], &in->buffer.b);

        // Complete hash
        out->result.t.size
            = CryptGetHashDigestSize(
                CryptGetContextAlg(&hashObject->state.hashState[0]));

        CryptCompleteHash2B(&hashObject->state.hashState[0], &out->result.b);

        // Check if the first block of the sequence has been received
        if(hashObject->attributes.firstBlock == CLEAR)
        {
            // If not, then this is the first block so see if it is 'safe'
            // to sign.
            if(TicketIsSafe(&in->buffer.b))
                hashObject->attributes.ticketSafe = SET;
        }

        // Output ticket
        out->validation.tag = TPM_ST_HASHCHECK;
        out->validation.hierarchy = in->hierarchy;
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
54
55          if(in->hierarchy == TPM_RH_NULL)
56          {
57              // Ticket is not required
58              out->validation.digest.t.size = 0;
59          }
60          else if(object->attributes.ticketSafe == CLEAR)
61          {
62              // Ticket is not safe to generate
63              out->validation.hierarchy = TPM_RH_NULL;
64              out->validation.digest.t.size = 0;
65          }
66          else
67          {
68              // Compute ticket
69              TicketComputeHashCheck(out->validation.hierarchy, hashAlg,
70                                     &out->result, &out->validation);
71          }
72      }
73      else
74      {
75          HASH_OBJECT     *hashObject = (HASH_OBJECT *)object;
76
77          //   Update last piece of data
78          CryptUpdateDigest2B(&hashObject->state.hmacState, &in->buffer.b);
79          // Complete hash/HMAC
80          out->result.t.size =
81              CryptGetHashDigestSize(
82                  CryptGetContextAlg(&hashObject->state.hmacState.hashState));
83          CryptCompleteHMAC2B(&(hashObject->state.hmacState), &out->result.b);
84
85          // No ticket is generated for HMAC sequence
86          out->validation.tag = TPM_ST_HASHCHECK;
87          out->validation.hierarchy = TPM_RH_NULL;
88          out->validation.digest.t.size = 0;
89      }
90
91  // Internal Data Update
92
93      // mark sequence object as evict so it will be flushed on the way out
94      object->attributes.evict = SET;
95
96      return TPM_RC_SUCCESS;
97  }
98  #endif // CC_SequenceComplete
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

18.6 TPM2_EventSequenceComplete

18.6.1 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

18.6.2 Command and Response

Table 103 — TPM2_EventSequenceComplete Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 104 — TPM2_EventSequenceComplete Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPML_DIGEST_VALUES	results	list of digests computed for the PCR

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

18.6.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "EventSequenceComplete_fp.h"
#ifdef TPM_CC_EventSequenceComplete  // Conditional expansion of this file
```

Table 105 — TPM2_EventSequenceComplete Errors

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

```
TPM_RC
TPM2_EventSequenceComplete(
    EventSequenceComplete_In    *in,            // IN: input parameter list
    EventSequenceComplete_Out   *out            // OUT: output parameter list
    )
{
    TPM_RC              result;
    HASH_OBJECT         *hashObject;
    UINT32              i;
    TPM_ALG_ID          hashAlg;

// Input validation

    // get the event sequence object pointer
    hashObject = (HASH_OBJECT *)ObjectGet(in->sequenceHandle);

    // input handle must reference an event sequence object
    if(hashObject->attributes.eventSeq != SET)
        return TPM_RC_MODE + RC_EventSequenceComplete_sequenceHandle;

    // see if a PCR extend is requested in call
    if(in->pcrHandle != TPM_RH_NULL)
    {
        // see if extend of the PCR is allowed at the locality of the command,
        if(!PCRIsExtendAllowed(in->pcrHandle))
            return TPM_RC_LOCALITY;
        // if an extend is going to take place, then check to see if there has
        // been an orderly shutdown. If so, and the selected PCR is one of the
        // state saved PCR, then the orderly state has to change. The orderly state
        // does not change for PCR that are not preserved.
        // NOTE: This doesn't just check for Shutdown(STATE) because the orderly
        // state will have to change if this is a state-saved PCR regardless
        // of the current state. This is because a subsequent Shutdown(STATE) will
        // check to see if there was an orderly shutdown and not do anything if
        // there was. So, this must indicate that a future Shutdown(STATE) has
        // something to do.
        if(gp.orderlyState != SHUTDOWN_NONE && PCRIsStateSaved(in->pcrHandle))
        {
            result = NvIsAvailable();
            if(result != TPM_RC_SUCCESS) return result;
            g_clearOrderly = TRUE;
        }
    }

// Command Output

    out->results.count = 0;

    for(i = 0; i < HASH_COUNT; i++)
    {
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

```
54          hashAlg = CryptGetHashAlgByIndex(i);
55          // Update last piece of data
56          CryptUpdateDigest2B(&hashObject->state.hashState[i], &in->buffer.b);
57          // Complete hash
58          out->results.digests[out->results.count].hashAlg = hashAlg;
59          CryptCompleteHash(&hashObject->state.hashState[i],
60                            CryptGetHashDigestSize(hashAlg),
61                            (BYTE *) &out->results.digests[out->results.count].digest);
62
63          // Extend PCR
64          if(in->pcrHandle != TPM_RH_NULL)
65              PCRExtend(in->pcrHandle, hashAlg,
66                        CryptGetHashDigestSize(hashAlg),
67                        (BYTE *) &out->results.digests[out->results.count].digest);
68          out->results.count++;
69      }
70
71  // Internal Data Update
72
73      // mark sequence object as evict so it will be flushed on the way out
74      hashObject->attributes.evict = SET;
75
76      return TPM_RC_SUCCESS;
77  }
78  #endif // CC_EventSequenceComplete
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

19 Attestation Commands

19.1 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.

All signing commands include a parameter (typically *inScheme*) for the caller to specify 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 or the key handle is TPM_RH_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 or the key handle is TPM_RH_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. If the caller sets *scheme* to TPM_ALG_NULL, then the default scheme of the key is used. For a restricted signing key, the key's scheme cannot be TPM_ALG_NULL and cannot be overridden.

EXAMPLE TPM_ALG_RSAPSS cannot be selected for an ECC key, because the scheme is compatible with the family of the key.

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 necssary to access the data that might be in an attestation structure.

NOTE 2 When *signHandle* is TPM_RH_NULL, *scheme* is still needs to be a valid signing scheme (can 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 32.16. The remaining attestation commands are collected in the remainder of clause 19.

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 *firmwareV*ersion numbers are obfuscated when the signing key is not in the Endorsement or Platform hierarchies.

The obfuscation value is computed by:

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

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. (See ISO/IEC 11889-1, clause 5.4, "KDF Label Parameters" for normative KDF label values.)

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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

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 ECDAA, the *qualifyingData* is used in a different manner (for details, see ISO/IEC 11889-1, Annex B.4.2, “ECDAA”).

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

19.2 TPM2_Certify

19.2.1 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 19.1 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. This indicates that the policy that is being used is a policy that is for certification, and not a policy that would approve another use. That is, authority to use an object does not grant authority to certify the object.

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 might 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.

NOTE 3 If *signHandle* is TPM_RH_NULL, the TPMS_ATTEST structure is returned and *signature* is a NULL Signature.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

19.2.2 Command and Response

Table 106 — TPM2_Certify Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 *scheme* for *signHandle* is TPM_ALG_NULL

Table 107 — TPM2_Certify Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	.
TPM2B_ATTEST	certifyInfo	the structure that was signed
TPMT_SIGNATURE	signature	the asymmetric signature over *certifyInfo* using the key referenced by *signHandle*

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

19.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Attest_spt_fp.h"
#include "Certify_fp.h"
#ifdef TPM_CC_Certify  // Conditional expansion of this file
```

Table 108 — TPM2_Certify Errors

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

```
TPM_RC
TPM2_Certify(
    Certify_In      *in,            // IN: input parameter list
    Certify_Out     *out            // OUT: output parameter list
    )
{
    TPM_RC                  result;
    TPMS_ATTEST             certifyInfo;

// Command Output

    // Filling in attest information
    // Common fields
    result = FillInAttestInfo(in->signHandle,
                              &in->inScheme,
                              &in->qualifyingData,
                              &certifyInfo);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_KEY)
            return TPM_RC_KEY + RC_Certify_signHandle;
        else
            return RcSafeAddToResult(result, RC_Certify_inScheme);
    }
    // Certify specific fields
    // Attestation type
    certifyInfo.type = TPM_ST_ATTEST_CERTIFY;
    // Certified object name
    certifyInfo.attested.certify.name.t.size =
        ObjectGetName(in->objectHandle,
                      &certifyInfo.attested.certify.name.t.name);
    // Certified object qualified name
    ObjectGetQualifiedName(in->objectHandle,
                           &certifyInfo.attested.certify.qualifiedName);

    // Sign attestation structure.  A NULL signature will be returned if
    // signHandle is TPM_RH_NULL.  A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
    // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned
    // by SignAttestInfo()
    result = SignAttestInfo(in->signHandle,
                            &in->inScheme,
                            &certifyInfo,
                            &in->qualifyingData,
                            &out->certifyInfo,
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
                                &out->signature);

    // TPM_RC_ATTRIBUTES cannot be returned here as FillInAttestInfo would already
    // have returned TPM_RC_KEY
    pAssert(result != TPM_RC_ATTRIBUTES);

    if(result != TPM_RC_SUCCESS)
        return result;

    // orderly state should be cleared because of the reporting of clock info
    // if signing happens
    if(in->signHandle != TPM_RH_NULL)
        g_clearOrderly = TRUE;

    return TPM_RC_SUCCESS;
}
#endif // CC_Certify
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

19.3 TPM2_CertifyCreation

19.3.1 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 19.1 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:

$$\mathbf{HMAC}(proof, (\text{TPM_ST_CREATION} \,||\, objectHandle{\rightarrow}Name \,||\, creationHash)) \quad (4)$$

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().

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

19.3.2 Command and Response

Table 109 — TPM2_CertifyCreation Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 *scheme* for *signHandle* is TPM_ALG_NULL
TPMT_TK_CREATION	creationTicket	ticket produced by TPM2_Create() or TPM2_CreatePrimary()

Table 110 — TPM2_CertifyCreation Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ATTEST	certifyInfo	the structure that was signed
TPMT_SIGNATURE	signature	the signature over *certifyInfo*

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

19.3.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Attest_spt_fp.h"
#include "CertifyCreation_fp.h"
#ifdef TPM_CC_CertifyCreation  // Conditional expansion of this file
```

Table 111 — TPM2_CertifyCreation Errors

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

```
TPM_RC
TPM2_CertifyCreation(
    CertifyCreation_In      *in,            // IN: input parameter list
    CertifyCreation_Out     *out            // OUT: output parameter list
    )
{
    TPM_RC                  result;
    TPM2B_NAME              name;
    TPMT_TK_CREATION        ticket;
    TPMS_ATTEST             certifyInfo;

// Input Validation

    // CertifyCreation specific input validation
    // Get certified object name
    name.t.size = ObjectGetName(in->objectHandle, &name.t.name);
    // Re-compute ticket
    TicketComputeCreation(in->creationTicket.hierarchy, &name,
                          &in->creationHash, &ticket);
    // Compare ticket
    if(!Memory2BEqual(&ticket.digest.b, &in->creationTicket.digest.b))
        return TPM_RC_TICKET + RC_CertifyCreation_creationTicket;

// Command Output
    // Common fields
    result = FillInAttestInfo(in->signHandle,  &in->inScheme, &in->qualifyingData,
                              &certifyInfo);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_KEY)
            return TPM_RC_KEY + RC_CertifyCreation_signHandle;
        else
            return RcSafeAddToResult(result, RC_CertifyCreation_inScheme);
    }

    // CertifyCreation specific fields
    // Attestation type
    certifyInfo.type = TPM_ST_ATTEST_CREATION;
    certifyInfo.attested.creation.objectName = name;

    // Copy the creationHash
    certifyInfo.attested.creation.creationHash = in->creationHash;
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
47
48      // Sign attestation structure.  A NULL signature will be returned if
49      // signHandle is TPM_RH_NULL.  A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
50      // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned at
51      // this point
52      result = SignAttestInfo(in->signHandle,
53                              &in->inScheme,
54                              &certifyInfo,
55                              &in->qualifyingData,
56                              &out->certifyInfo,
57                              &out->signature);
58
59      // TPM_RC_ATTRIBUTES cannot be returned here as FillInAttestInfo would already
60      // have returned TPM_RC_KEY
61      pAssert(result != TPM_RC_ATTRIBUTES);
62
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  }
73  #endif // CC_CertifyCreation
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

19.4 TPM2_Quote

19.4.1 General Description

This command is used to quote PCR values.

NOTE See 19.1 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 specified in ISO/IEC 11889-1, clause 17.5, “Selecting Multiple PCR”.

NOTE 2 If *signHandle* is TPM_RH_NULL, the TPMS_ATTEST structure is returned and *signature* is a NULL Signature.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

19.4.2 Command and Response

Table 112 — TPM2_Quote Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 *scheme* for *signHandle* is TPM_ALG_NULL
TPML_PCR_SELECTION	PCRselect	PCR set to quote

Table 113 — TPM2_Quote Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ATTEST	quoted	the quoted information
TPMT_SIGNATURE	signature	the signature over *quoted*

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

19.4.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Attest_spt_fp.h"
#include "Quote_fp.h"
#ifdef TPM_CC_Quote  // Conditional expansion of this file
```

Table 114 — TPM2_Quote Errors

Error Returns	Meaning
TPM_RC_KEY	*signHandle* 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

```
TPM_RC
TPM2_Quote(
    Quote_In        *in,            // IN: input parameter list
    Quote_Out       *out            // OUT: output parameter list
    )
{
    TPM_RC                  result;
    TPMI_ALG_HASH           hashAlg;
    TPMS_ATTEST             quoted;

// Command Output

    // Filling in attest information
    // Common fields
    // FillInAttestInfo may return TPM_RC_SCHEME or TPM_RC_KEY
    result = FillInAttestInfo(in->signHandle,
                              &in->inScheme,
                              &in->qualifyingData,
                              &quoted);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_KEY)
            return TPM_RC_KEY + RC_Quote_signHandle;
        else
            return RcSafeAddToResult(result, RC_Quote_inScheme);
    }

    // Quote specific fields
    // Attestation type
    quoted.type = TPM_ST_ATTEST_QUOTE;

    // Get hash algorithm in sign scheme.  This hash algorithm is used to
    // compute PCR digest. If there is no algorithm, then the PCR cannot
    // be digested and this command returns TPM_RC_SCHEME
    hashAlg = in->inScheme.details.any.hashAlg;

    if(hashAlg == TPM_ALG_NULL)
        return TPM_RC_SCHEME + RC_Quote_inScheme;

    // Compute PCR digest
    PCRComputeCurrentDigest(hashAlg,
                            &in->PCRselect,
                            &quoted.attested.quote.pcrDigest);

    // Copy PCR select.  "PCRselect" is modified in PCRComputeCurrentDigest
    // function
    quoted.attested.quote.pcrSelect = in->PCRselect;
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
52
53      // Sign attestation structure.  A NULL signature will be returned if
54      // signHandle is TPM_RH_NULL.  TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES
55      // error may be returned by SignAttestInfo.
56      // NOTE: TPM_RC_ATTRIBUTES means that the key is not a signing key but that
57      // was checked above and TPM_RC_KEY was returned. TPM_RC_VALUE means that the
58      // value to sign is too large but that means that the digest is too big and
59      // that can't happen.
60      result = SignAttestInfo(in->signHandle,
61                              &in->inScheme,
62                              &quoted,
63                              &in->qualifyingData,
64                              &out->quoted,
65                              &out->signature);
66      if(result != TPM_RC_SUCCESS)
67          return result;
68
69      // orderly state should be cleared because of the reporting of clock info
70      // if signing happens
71      if(in->signHandle != TPM_RH_NULL)
72          g_clearOrderly = TRUE;
73
74      return TPM_RC_SUCCESS;
75  }
76  #endif // CC_Quote
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

19.5 TPM2_GetSessionAuditDigest

19.5.1 General Description

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

NOTE 1 See 19.1 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 Endorsement Authorization) 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 If sessionHandle is used as an audit session for this command, the command is audited in the same manner as any other command.

NOTE 4 If *signHandle* is TPM_RH_NULL, the TPMS_ATTEST structure is returned and *signature* is a NULL Signature.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

19.5.2 Command and Response

Table 115 — TPM2_GetSessionAuditDigest Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 *scheme* for *signHandle* is TPM_ALG_NULL

Table 116 — TPM2_GetSessionAuditDigest Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ATTEST	auditInfo	the audit information that was signed
TPMT_SIGNATURE	signature	the signature over *auditInfo*

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

19.5.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Attest_spt_fp.h"
#include "GetSessionAuditDigest_fp.h"
#ifdef TPM_CC_GetSessionAuditDigest  // Conditional expansion of this file
```

Table 117 — TPM2_GetSessionAuditDigest Errors

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

```
TPM_RC
TPM2_GetSessionAuditDigest(
    GetSessionAuditDigest_In    *in,            // IN: input parameter list
    GetSessionAuditDigest_Out   *out            // OUT: output parameter list
    )
{
    TPM_RC                  result;
    SESSION                *session;
    TPMS_ATTEST             auditInfo;

// Input Validation

    // SessionAuditDigest specific input validation
    // Get session pointer
    session = SessionGet(in->sessionHandle);

    // session must be an audit session
    if(session->attributes.isAudit == CLEAR)
        return TPM_RC_TYPE + RC_GetSessionAuditDigest_sessionHandle;

// Command Output

    // Filling in attest information
    // Common fields
    result = FillInAttestInfo(in->signHandle,
                              &in->inScheme,
                              &in->qualifyingData,
                              &auditInfo);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_KEY)
            return TPM_RC_KEY + RC_GetSessionAuditDigest_signHandle;
        else
            return RcSafeAddToResult(result, RC_GetSessionAuditDigest_inScheme);
    }

    // SessionAuditDigest specific fields
    // Attestation type
    auditInfo.type = TPM_ST_ATTEST_SESSION_AUDIT;

```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

```
45          // Copy digest
46          auditInfo.attested.sessionAudit.sessionDigest = session->u2.auditDigest;
47
48          // Exclusive audit session
49          if(g_exclusiveAuditSession == in->sessionHandle)
50              auditInfo.attested.sessionAudit.exclusiveSession = TRUE;
51          else
52              auditInfo.attested.sessionAudit.exclusiveSession = FALSE;
53
54          // Sign attestation structure.  A NULL signature will be returned if
55          // signHandle is TPM_RH_NULL.  A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
56          // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned at
57          // this point
58          result = SignAttestInfo(in->signHandle,
59                                  &in->inScheme,
60                                  &auditInfo,
61                                  &in->qualifyingData,
62                                  &out->auditInfo,
63                                  &out->signature);
64          if(result != TPM_RC_SUCCESS)
65              return result;
66
67          // orderly state should be cleared because of the reporting of clock info
68          // if signing happens
69          if(in->signHandle != TPM_RH_NULL)
70              g_clearOrderly = TRUE;
71
72          return TPM_RC_SUCCESS;
73      }
74      #endif // CC_GetSessionAuditDigest
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

19.6 TPM2_GetCommandAuditDigest

19.6.1 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 19.1 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. If signHandle is TPM_RH_NULL, *signature* is the Empty Buffer and the audit digest is not 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 Endorsement Authorization) as well as authorization to use the key associated with *signHandle*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

19.6.2 Command and Response

Table 118 — TPM2_GetCommandAuditDigest Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 *scheme* for *signHandle* is TPM_ALG_NULL

Table 119 — TPM2_GetCommandAuditDigest Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_ATTEST	auditInfo	the auditInfo that was signed
TPMT_SIGNATURE	signature	the signature over *auditInfo*

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

19.6.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Attest_spt_fp.h"
#include "GetCommandAuditDigest_fp.h"
#ifdef TPM_CC_GetCommandAuditDigest  // Conditional expansion of this file
```

Table 120 — TPM2_GetCommandAuditDigest Errors

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

```
TPM_RC
TPM2_GetCommandAuditDigest(
    GetCommandAuditDigest_In    *in,            // IN: input parameter list
    GetCommandAuditDigest_Out   *out            // OUT: output parameter list
    )
{
    TPM_RC                  result;
    TPMS_ATTEST             auditInfo;

// Command Output

    // Filling in attest information
    // Common fields
    result = FillInAttestInfo(in->signHandle,
                              &in->inScheme,
                              &in->qualifyingData,
                              &auditInfo);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_KEY)
            return TPM_RC_KEY + RC_GetCommandAuditDigest_signHandle;
        else
            return RcSafeAddToResult(result, RC_GetCommandAuditDigest_inScheme);
    }

    // CommandAuditDigest specific fields
    // Attestation type
    auditInfo.type = TPM_ST_ATTEST_COMMAND_AUDIT;

    // Copy audit hash algorithm
    auditInfo.attested.commandAudit.digestAlg = gp.auditHashAlg;

    // Copy counter value
    auditInfo.attested.commandAudit.auditCounter = gp.auditCounter;

    // Copy command audit log
    auditInfo.attested.commandAudit.auditDigest = gr.commandAuditDigest;
    CommandAuditGetDigest(&auditInfo.attested.commandAudit.commandDigest);

    // Sign attestation structure.  A NULL signature will be returned if
    // signHandle is TPM_RH_NULL.  A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
    // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned at
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
47          // this point
48          result = SignAttestInfo(in->signHandle,
49                                  &in->inScheme,
50                                  &auditInfo,
51                                  &in->qualifyingData,
52                                  &out->auditInfo,
53                                  &out->signature);
54
55          if(result != TPM_RC_SUCCESS)
56              return result;
57
58      // Internal Data Update
59
60          if(in->signHandle != TPM_RH_NULL)
61          {
62              // Reset log
63              gr.commandAuditDigest.t.size = 0;
64
65              // orderly state should be cleared because of the update in
66              // commandAuditDigest, as well as the reporting of clock info
67              g_clearOrderly = TRUE;
68          }
69
70          return TPM_RC_SUCCESS;
71      }
72      #endif // CC_GetCommandAuditDigest
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

19.7 TPM2_GetTime

19.7.1 General Description

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

NOTE 1 See 19.1 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 TPMS_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. This command requires Endorsement Authorization.

NOTE 3 If *signHandle* is TPM_RH_NULL, the TPMS_ATTEST structure is returned and *signature* is a NULL Signature.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

19.7.2 Command and Response

Table 121 — TPM2_GetTime Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 *keyHandle* 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 *scheme* for *signHandle* is TPM_ALG_NULL

Table 122 — TPM2_GetTime Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	.
TPM2B_ATTEST	timeInfo	standard TPM-generated attestation block
TPMT_SIGNATURE	signature	the signature over *timeInfo*

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

19.7.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Attest_spt_fp.h"
#include "GetTime_fp.h"
#ifdef TPM_CC_GetTime  // Conditional expansion of this file
```

Table 123 — TPM2_GetTime Errors

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

```
TPM_RC
TPM2_GetTime(
    GetTime_In      *in,            // IN: input parameter list
    GetTime_Out     *out            // OUT: output parameter list
    )
{
    TPM_RC                  result;
    TPMS_ATTEST             timeInfo;

// Command Output

    // Filling in attest information
    // Common fields
    result = FillInAttestInfo(in->signHandle,
                              &in->inScheme,
                              &in->qualifyingData,
                              &timeInfo);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_KEY)
            return TPM_RC_KEY + RC_GetTime_signHandle;
        else
            return RcSafeAddToResult(result, RC_GetTime_inScheme);
    }

    // GetClock specific fields
    // Attestation type
    timeInfo.type = TPM_ST_ATTEST_TIME;

    // current clock in plain text
    timeInfo.attested.time.time.time = g_time;
    TimeFillInfo(&timeInfo.attested.time.time.clockInfo);

    // Firmware version in plain text
    timeInfo.attested.time.firmwareVersion
        = ((UINT64) gp.firmwareV1) << 32;
    timeInfo.attested.time.firmwareVersion += gp.firmwareV2;

    // Sign attestation structure.  A NULL signature will be returned if
    // signHandle is TPM_RH_NULL.  A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
    // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned at
    // this point
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
47      result = SignAttestInfo(in->signHandle,
48                              &in->inScheme,
49                              &timeInfo,
50                              &in->qualifyingData,
51                              &out->timeInfo,
52                              &out->signature);
53      if(result != TPM_RC_SUCCESS)
54          return result;
55
56      // orderly state should be cleared because of the reporting of clock info
57      // if signing happens
58      if(in->signHandle != TPM_RH_NULL)
59          g_clearOrderly = TRUE;
60
61      return TPM_RC_SUCCESS;
62  }
63  #endif // CC_GetTime
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

20 Ephemeral EC Keys

20.1 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. 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.

EXAMPLE An example of a single-use key used in the command that creates it is TPM2_Duplicate() where an ephemeral key is created for a single pass key exchange with another TPM.

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.

EXAMPLE A 128 bit array would allow 128 keys to be "pending".

The management of the array is specified in greater detail in ISO/IEC 11889-1, Annex B.2, “Split Operations”.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

20.2 TPM2_Commit

20.2.1 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) and the signing scheme must be anonymous (TPM_RC_SCHEME). Currently, TPM_ALG_ECDAA is the only defined anonymous scheme.

NOTE This command cannot be used with a sign+decrypt key because that type of key is required to have a scheme of TPM_ALG_NULL.

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.

The algorithm is specified in ISO/IEC 11889-1, Annex B.2.3, “TPM2_Commit()”.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

20.2.2 Command and Response

Table 124 — TPM2_Commit Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 *signHandle*
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 125 — TPM2_Commit Response

Type	Name	Description
TPM_ST	tag	see 7
UINT32	paramSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	K	ECC point $K := [d_s](x2, y2)$
TPM2B_ECC_POINT	L	ECC point $L := [r](x2, y2)$
TPM2B_ECC_POINT	E	ECC point $E := [r]P1$
UINT16	counter	least-significant 16 bits of *commitCount*

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

20.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Commit_fp.h"
#ifdef TPM_CC_Commit  // Conditional expansion of this file
#ifdef TPM_ALG_ECC
```

Table 126 — TPM2_Commit Response Errors

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

```
TPM_RC
TPM2_Commit(
    Commit_In       *in,            // IN: input parameter list
    Commit_Out      *out            // OUT: output parameter list
    )
{
    OBJECT                  *eccKey;
    TPMS_ECC_POINT           P2;
    TPMS_ECC_POINT          *pP2 = NULL;
    TPMS_ECC_POINT          *pP1 = NULL;
    TPM2B_ECC_PARAMETER      r;
    TPM2B                   *p;
    TPM_RC                   result;
    TPMS_ECC_PARMS          *parms;

// Input Validation

    eccKey = ObjectGet(in->signHandle);
    parms = & eccKey->publicArea.parameters.eccDetail;

    // Input key must be an ECC key
    if(eccKey->publicArea.type != TPM_ALG_ECC)
        return TPM_RC_KEY + RC_Commit_signHandle;

    // This command may only be used with a sign-only key using an anonymous
    // scheme.
    // NOTE: a sign + decrypt key has no scheme so it will not be an anonymous one
    // and an unrestricted sign key might no have a signing scheme but it can't
    // be use in Commit()
   if(!CryptIsSchemeAnonymous(parms->scheme.scheme))
            return TPM_RC_SCHEME + RC_Commit_signHandle;

    // Make sure that both parts of P2 are present if either is present
    if((in->s2.t.size == 0) != (in->y2.t.size == 0))
        return TPM_RC_SIZE + RC_Commit_y2;

    // Get prime modulus for the curve. This is needed later but getting this now
    // allows confirmation that the curve exists
    p = (TPM2B *)CryptEccGetParameter('p', parms->curveID);

```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
45      // if no p, then the curve ID is bad
46      // NOTE: This should never occur if the input unmarshaling code is working
47      // correctly
48      pAssert(p != NULL);
49
50      // Get the random value that will be used in the point multiplications
51      // Note: this does not commit the count.
52      if(!CryptGenerateR(&r, NULL, parms->curveID, &eccKey->name))
53          return TPM_RC_NO_RESULT;
54
55      // Set up P2 if s2 and Y2 are provided
56      if(in->s2.t.size != 0)
57      {
58          pP2 = &P2;
59
60          // copy y2 for P2
61          MemoryCopy2B(&P2.y.b, &in->y2.b, sizeof(P2.y.t.buffer));
62          // Compute x2  HnameAlg(s2) mod p
63
64          //      do the hash operation on s2 with the size of curve 'p'
65          P2.x.t.size = CryptHashBlock(eccKey->publicArea.nameAlg,
66                                       in->s2.t.size,
67                                       in->s2.t.buffer,
68                                       p->size,
69                                       P2.x.t.buffer);
70
71          // If there were error returns in the hash routine, indicate a problem
72          // with the hash in
73          if(P2.x.t.size == 0)
74              return TPM_RC_HASH + RC_Commit_signHandle;
75
76          //  set p2.x = hash(s2) mod p
77          if(CryptDivide(&P2.x.b, p, NULL, &P2.x.b) != TPM_RC_SUCCESS)
78              return TPM_RC_NO_RESULT;
79
80          if(!CryptEccIsPointOnCurve(parms->curveID, pP2))
81              return TPM_RC_ECC_POINT + RC_Commit_s2;
82
83          if(eccKey->attributes.publicOnly == SET)
84              return TPM_RC_KEY + RC_Commit_signHandle;
85
86      }
87      // If there is a P1, make sure that it is on the curve
88      // NOTE: an "empty" point has two UINT16 values which are the size values
89      // for each of the coordinates.
90      if(in->P1.t.size > 4)
91      {
92          pP1 = &in->P1.t.point;
93          if(!CryptEccIsPointOnCurve(parms->curveID, pP1))
94              return TPM_RC_ECC_POINT + RC_Commit_P1;
95      }
96
97      // Pass the parameters to CryptCommit.
98      // The work is not done in-line because it does several point multiplies
99      // with the same curve.  There is significant optimization by not
100     // having to reload the curve parameters multiple times.
101     result = CryptCommitCompute(&out->K.t.point,
102                                 &out->L.t.point,
103                                 &out->E.t.point,
104                                 parms->curveID,
105                                 pP1,
106                                 pP2,
107                                 &eccKey->sensitive.sensitive.ecc,
108                                 &r);
109     if(result != TPM_RC_SUCCESS)
110         return result;
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
111
112         out->K.t.size = TPMS_ECC_POINT_Marshal(&out->K.t.point, NULL, NULL);
113         out->L.t.size = TPMS_ECC_POINT_Marshal(&out->L.t.point, NULL, NULL);
114         out->E.t.size = TPMS_ECC_POINT_Marshal(&out->E.t.point, NULL, NULL);
115
116         // The commit computation was successful so complete the commit by setting
117         // the bit
118         out->counter = CryptCommit();
119
120         return TPM_RC_SUCCESS;
121     }
122     #endif
123     #endif // CC_Commit
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

20.3 TPM2_EC_Ephemeral

20.3.1 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*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

20.3.2 Command and Response

Table 127 — TPM2_EC_Ephemeral Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or encrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	paramSize	
TPM_CC	commandCode	TPM_CC_EC_Ephemeral
TPMI_ECC_CURVE	curveID	The curve for the computed ephemeral point

Table 128 — TPM2_EC_Ephemeral Response

Type	Name	Description
TPM_ST	tag	see 7
UINT32	paramSize	
TPM_RC	responseCode	
TPM2B_ECC_POINT	Q	ephemeral public key $Q := [r]G$
UINT16	counter	least-significant 16 bits of *commitCount*

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

20.3.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "EC_Ephemeral_fp.h"
#ifdef TPM_CC_EC_Ephemeral  // Conditional expansion of this file
#ifdef TPM_ALG_ECC

TPM_RC
TPM2_EC_Ephemeral(
    EC_Ephemeral_In     *in,            // IN: input parameter list
    EC_Ephemeral_Out    *out            // OUT: output parameter list
    )
{
    TPM2B_ECC_PARAMETER      r;

    // Get the random value that will be used in the point multiplications
    // Note: this does not commit the count.
    if(!CryptGenerateR(&r,
                       NULL,
                       in->curveID,
                       NULL))
        return TPM_RC_NO_RESULT;

    CryptEccPointMultiply(&out->Q.t.point, in->curveID, &r, NULL);

    // commit the count value
    out->counter = CryptCommit();

    return TPM_RC_SUCCESS;
}
#endif
#endif // CC_EC_Ephemeral
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

21 Signing and Signature Verification

21.1 TPM2_VerifySignature

21.1.1 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 might 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 needed to allow verification of the symmetric signature (the HMAC).

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

21.1.2 Command and Response

Table 129 — TPM2_VerifySignature Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or encrypt session is present; otherwise, TPM_ST_NO_SESSIONS
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 130 — TPM2_VerifySignature Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPMT_TK_VERIFIED	validation	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

21.1.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "VerifySignature_fp.h"
#ifdef TPM_CC_VerifySignature  // Conditional expansion of this file
```

Table 131 — TPM2_VerifySignature Errors

Error Returns	Meaning
TPM_RC_ATTRIBUTES	*keyHandle* does not reference a signing key
TPM_RC_SIGNATURE	signature is not genuine
TPM_RC_SCHEME	CryptVerifySignature()
TPM_RC_HANDLE	the input handle is references an HMAC key but the private portion is not loaded

```
TPM_RC
TPM2_VerifySignature(
    VerifySignature_In      *in,            // IN: input parameter list
    VerifySignature_Out     *out            // OUT: output parameter list
    )
{
    TPM_RC                   result;
    TPM2B_NAME               name;
    OBJECT                  *signObject;
    TPMI_RH_HIERARCHY        hierarchy;

// Input Validation

    // Get sign object pointer
    signObject = ObjectGet(in->keyHandle);

    // The object to validate the signature must be a signing key.
    if(signObject->publicArea.objectAttributes.sign != SET)
        return TPM_RC_ATTRIBUTES + RC_VerifySignature_keyHandle;

    // Validate Signature.  TPM_RC_SCHEME, TPM_RC_HANDLE or TPM_RC_SIGNATURE
    // error may be returned by CryptCVerifySignatrue()
    result = CryptVerifySignature(in->keyHandle, &in->digest, &in->signature);
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, RC_VerifySignature_signature);

// Command Output

    hierarchy = ObjectGetHierarchy(in->keyHandle);
    if(   hierarchy == TPM_RH_NULL
       || signObject->publicArea.nameAlg == TPM_ALG_NULL)
    {
        // produce empty ticket if hierarchy is TPM_RH_NULL or nameAlg is
        // TPM_ALG_NULL
        out->validation.tag = TPM_ST_VERIFIED;
        out->validation.hierarchy = TPM_RH_NULL;
        out->validation.digest.t.size = 0;
    }
    else
    {
        // Get object name that verifies the signature
        name.t.size = ObjectGetName(in->keyHandle, &name.t.name);
        // Compute ticket
        TicketComputeVerified(hierarchy, &in->digest, &name, &out->validation);
    }
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
49
50          return TPM_RC_SUCCESS;
51      }
52      #endif // CC_VerifySignature
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

21.2 TPM2_Sign

21.2.1 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 the TPM HMAC commands.

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 uses a hash algorithm, the algorithm is defined in the qualifying data of the scheme. This is the same algorithm that is required to be used in producing *digest*. The size of *digest* needs to match that of the hash algorithm in 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.

If *validation* is provided, then the hash algorithm used in computing the digest is required to be the hash algorithm specified in the scheme of *keyHandle* (TPM_RC_TICKET).

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.

NOTE 4 If *keyHandle* is both a sign and decrypt key, *keyHandle* will have an scheme of TPM_ALG_NULL. If *validation* is provided, then it needs to be a NULL validation ticket or the ticket validation will fail.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

21.2.2 Command and Response

Table 132 — TPM2_Sign Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 *scheme* for *keyHandle* is TPM_ALG_NULL
TPMT_TK_HASHCHECK	validation	proof that digest was created by the TPM If *keyHandle* is not a restricted signing key, then this may be a NULL Ticket with *tag* = TPM_ST_CHECKHASH.

Table 133 — TPM2_Sign Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPMT_SIGNATURE	signature	the signature

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

21.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Sign_fp.h"
#ifdef TPM_CC_Sign  // Conditional expansion of this file
#include "Attest_spt_fp.h"
```

Table 134 — TPM2_Sign Response Errors

Error Returns	Meaning
TPM_RC_BINDING	The public and private portions of the key are not properly bound.
TPM_RC_KEY	*signHandle* 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
TPM_RC_TICKET	*validation* is not a valid ticket
TPM_RC_VALUE	the value to sign is larger than allowed for the type of *keyHandle*

```
TPM_RC
TPM2_Sign(
    Sign_In         *in,            // IN: input parameter list
    Sign_Out        *out            // OUT: output parameter list
    )
{
    TPM_RC                  result;
    TPMT_TK_HASHCHECK       ticket;
    OBJECT                  *signKey;

// Input Validation
    // Get sign key pointer
    signKey = ObjectGet(in->keyHandle);

    // pick a scheme for sign.  If the input sign scheme is not compatible with
    // the default scheme, return an error.
    result = CryptSelectSignScheme(in->keyHandle, &in->inScheme);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_KEY)
            return TPM_RC_KEY + RC_Sign_keyHandle;
        else
            return RcSafeAddToResult(result, RC_Sign_inScheme);
    }

    // If validation is provided, or the key is restricted, check the ticket
    if(   in->validation.digest.t.size != 0
       || signKey->publicArea.objectAttributes.restricted == SET)
    {
        // Compute and compare ticket
        TicketComputeHashCheck(in->validation.hierarchy,
                               in->inScheme.details.any.hashAlg,
                               &in->digest, &ticket);

        if(!Memory2BEqual(&in->validation.digest.b, &ticket.digest.b))
            return TPM_RC_TICKET + RC_Sign_validation;
    }
    else
    // If we don't have a ticket, at least verify that the provided 'digest'
    // is the size of the scheme hashAlg digest.
    // NOTE: this does not guarantee that the 'digest' is actually produced using
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
46      // the indicated hash algorithm, but at least it might be.
47      {
48          if(     in->digest.t.size
49              !=  CryptGetHashDigestSize(in->inScheme.details.any.hashAlg))
50              return TPM_RCS_SIZE + RC_Sign_digest;
51      }
52
53  // Command Output
54      // Sign the hash. A TPM_RC_VALUE or TPM_RC_SCHEME
55      // error may be returned at this point
56      result = CryptSign(in->keyHandle, &in->inScheme, &in->digest, &out->signature);
57
58      return result;
59  }
60  #endif // CC_Sign
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

22 Command Audit

22.1 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:

$$commandAuditDigest_{new} := \mathbf{H}_{auditAlg}(commandAuditDigest_{old} \,||\, cpHash \,||\, rpHash) \quad (5)$$

where

$\boldsymbol{H}_{auditAlg}$	hash function using the algorithm of the audit sequence
$commandAuditDigest$	accumulated digest
$cpHash$	the command parameter hash
$rpHash$	the response parameter hash

auditAlg, the hash algorithm, is set using TPM2_SetCommandCodeAuditStatus.

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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

22.2 TPM2_SetCommandCodeAuditStatus

22.2.1 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 an audited event. If TPM_CC_SetCommandCodeAuditStatus is in *clearList,* the fact that it is in *clearList* 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 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).

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

22.2.2 Command and Response

Table 135 — TPM2_SetCommandCodeAuditStatus Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_SetCommandCodeAuditStatus {NV}
TPMI_RH_PROVISION	@auth	TPM_RH_OWNER 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 136 — TPM2_SetCommandCodeAuditStatus Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

22.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "SetCommandCodeAuditStatus_fp.h"
#ifdef TPM_CC_SetCommandCodeAuditStatus  // Conditional expansion of this file
TPM_RC
TPM2_SetCommandCodeAuditStatus(
    SetCommandCodeAuditStatus_In    *in             // IN: input parameter list
    )
{
    TPM_RC          result;
    UINT32          i;
    BOOL            changed = FALSE;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS)
        return result;

// Internal Data Update

    // Update hash algorithm
    if(   in->auditAlg != TPM_ALG_NULL
       && in->auditAlg != gp.auditHashAlg)
    {
        // Can't change the algorithm and command list at the same time
        if(in->setList.count != 0 || in->clearList.count != 0)
            return TPM_RC_VALUE + RC_SetCommandCodeAuditStatus_auditAlg;

        // Change the hash algorithm for audit
        gp.auditHashAlg = in->auditAlg;

        // Set the digest size to a unique value that indicates that the digest
        // algorithm has been changed. The size will be cleared to zero in the
        // command audit processing on exit.
        gr.commandAuditDigest.t.size = 1;

        // Save the change of command audit data (this sets g_updateNV so that NV
        // will be updated on exit.)
        NvWriteReserved(NV_AUDIT_HASH_ALG, &gp.auditHashAlg);

    } else {

        // Process set list
        for(i = 0; i < in->setList.count; i++)

            // If change is made in CommandAuditSet, set changed flag
            if(CommandAuditSet(in->setList.commandCodes[i]))
                changed = TRUE;

        // Process clear list
        for(i = 0; i < in->clearList.count; i++)
            // If change is made in CommandAuditClear, set changed flag
            if(CommandAuditClear(in->clearList.commandCodes[i]))
                changed = TRUE;

        // if change was made to command list, update NV
        if(changed)
            // this sets g_updateNV so that NV will be updated on exit.
            NvWriteReserved(NV_AUDIT_COMMANDS, &gp.auditComands);
    }

```

IECNORM.COM Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

```
63          return TPM_RC_SUCCESS;
64      }
65      #endif // CC_SetCommandCodeAuditStatus
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23 Integrity Collection (PCR)

23.1 Introduction

In ISO/IEC 11889 (first edition), an Event was hashed using SHA-1 and then the 20-octet digest was extended to a PCR using TPM_Extend(). ISO/IEC 11889 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 specified 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 can 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 might not represent the trust state of the platform.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

23.2 TPM2_PCR_Extend

23.2.1 General Description

This command is used to cause an update to the indicated PCR. The *digests* parameter contains one or more tagged digest values 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 the 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:

$$PCR.digest_{new}\,[pcrNum][alg] \coloneqq \mathbf{H}_{alg}(PCR.digest_{old}\,[pcrNum][alg] \,||\, data[alg].buffer)) \quad (6)$$

where

$\mathbf{H}_{alg}()$	hash function using the hash algorithm associated with the PCR instance
$PCR.digest$	the digest value in a PCR
$pcrNum$	the PCR numeric selector (*pcrHandle*)
alg	the PCR algorithm selector for the digest
$data[alg].buffer$	the bank-specific data to be extended

If no digest value is specified for a bank, then the PCR in that bank is 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. This permits the caller to probe for implemented hash algorithms as an alternative to TPM2_GetCapability..

NOTE 3 This command allows a list of digests so that PCR in all banks can 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.2.2 Command and Response

Table 137 — TPM2_PCR_Extend Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 138 — TPM2_PCR_Extend Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PCR_Extend_fp.h"
#ifdef TPM_CC_PCR_Extend  // Conditional expansion of this file
```

Table 139 — TPM2_PCR_Extend Errors

Error Returns	Meaning
TPM_RC_LOCALITY	current command locality is not allowed to extend the PCR referenced by *pcrHandle*

```
TPM_RC
TPM2_PCR_Extend(
    PCR_Extend_In   *in             // IN: input parameter list
    )
{
    TPM_RC              result;
    UINT32              i;

// Input Validation

    // NOTE: This function assumes that the unmarshaling function for 'digests' will
    // have validated that all of the indicated hash algorithms are valid. If the
    // hash algorithms are correct, the unmarshaling code will unmarshal a digest
    // of the size indicated by the hash algorithm. If the overall size is not
    // consistent, the unmarshaling code will run out of input data or have input
    // data left over. In either case, it will cause an unmarshaling error and this
    // function will not be called.

    // For NULL handle, do nothing and return success
    if(in->pcrHandle == TPM_RH_NULL)
        return TPM_RC_SUCCESS;

    // Check if the extend operation is allowed by the current command locality
    if(!PCRIsExtendAllowed(in->pcrHandle))
        return TPM_RC_LOCALITY;

    // If PCR is state saved and we need to update orderlyState, check NV
    // availability
    if(PCRIsStateSaved(in->pcrHandle) && gp.orderlyState != SHUTDOWN_NONE)
    {
        result = NvIsAvailable();
        if(result != TPM_RC_SUCCESS) return result;
        g_clearOrderly = TRUE;
    }

// Internal Data Update

    // Iterate input digest list to extend
    for(i = 0; i < in->digests.count; i++)
    {
        PCRExtend(in->pcrHandle, in->digests.digests[i].hashAlg,
                  CryptGetHashDigestSize(in->digests.digests[i].hashAlg),
                  (BYTE *) &in->digests.digests[i].digest);
    }

    return TPM_RC_SUCCESS;
}
#endif // CC_PCR_Extend
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.3 TPM2_PCR_Event

23.3.1 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, the *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. If *pcrHandle* is TPM_RH_NULL, the TPM may return either an empty list or a digest for each bank.

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* can contain either a SHA1 and a SHA256 digest or just a SHA1 digest.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

23.3.2 Command and Response

Table 140 — TPM2_PCR_Event Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 141 — TPM2_PCR_Event Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	.
TPML_DIGEST_VALUES	digests	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.3.3 Detailed Actions

```
1   #include "InternalRoutines.h"
2   #include "PCR_Event_fp.h"
3   #ifdef TPM_CC_PCR_Event  // Conditional expansion of this file
```

Table 142 — TPM2_PCR_Event Errors

Error Returns	Meaning
TPM_RC_LOCALITY	current command locality is not allowed to extend the PCR referenced by *pcrHandle*

```
4   TPM_RC
5   TPM2_PCR_Event(
6       PCR_Event_In    *in,            // IN: input parameter list
7       PCR_Event_Out   *out            // OUT: output parameter list
8       )
9   {
10      TPM_RC              result;
11      HASH_STATE          hashState;
12      UINT32              i;
13      UINT16              size;
14
15  // Input Validation
16
17      // If a PCR extend is required
18      if(in->pcrHandle != TPM_RH_NULL)
19      {
20          // If the PCR is not allow to extend, return error
21          if(!PCRIsExtendAllowed(in->pcrHandle))
22              return TPM_RC_LOCALITY;
23
24          // If PCR is state saved and we need to update orderlyState, check NV
25          // availability
26          if(PCRIsStateSaved(in->pcrHandle) && gp.orderlyState != SHUTDOWN_NONE)
27          {
28              result = NvIsAvailable();
29              if(result != TPM_RC_SUCCESS) return result;
30              g_clearOrderly = TRUE;
31          }
32      }
33
34  // Internal Data Update
35
36      out->digests.count = HASH_COUNT;
37
38      // Iterate supported PCR bank algorithms to extend
39      for(i = 0; i < HASH_COUNT; i++)
40      {
41          TPM_ALG_ID  hash = CryptGetHashAlgByIndex(i);
42          out->digests.digests[i].hashAlg = hash;
43          size = CryptStartHash(hash, &hashState);
44          CryptUpdateDigest2B(&hashState, &in->eventData.b);
45          CryptCompleteHash(&hashState, size,
46                            (BYTE *) &out->digests.digests[i].digest);
47          if(in->pcrHandle != TPM_RH_NULL)
48              PCRExtend(in->pcrHandle, hash, size,
49                        (BYTE *) &out->digests.digests[i].digest);
50      }
51
52      return TPM_RC_SUCCESS;
53  }
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
54      #endif // CC_PCR_Event
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.4 TPM2_PCR_Read

23.4.1 General Description

This command returns the values of all PCR specified in *pcrSelectionIn*.

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 ISO/IEC 11889-2 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 *pcrValues.*

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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

23.4.2 Command and Response

Table 143 — TPM2_PCR_Read Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PCR_Read
TPML_PCR_SELECTION	pcrSelectionIn	The selection of PCR to read

Table 144 — TPM2_PCR_Read Response

Type	Name	Description
TPM_ST	tag	see clause 7
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 *pcrSelect* as tagged digests

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.4.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PCR_Read_fp.h"
#ifdef TPM_CC_PCR_Read  // Conditional expansion of this file
TPM_RC
TPM2_PCR_Read(
    PCR_Read_In     *in,            // IN: input parameter list
    PCR_Read_Out    *out            // OUT: output parameter list
    )
{
// Command Output

    // Call PCR read function.  input pcrSelectionIn parameter could be changed
    // to reflect the actual PCR being returned
    PCRRead(&in->pcrSelectionIn, &out->pcrValues, &out->pcrUpdateCounter);

    out->pcrSelectionOut = in->pcrSelectionIn;

    return TPM_RC_SUCCESS;
}
#endif // CC_PCR_Read
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.5 TPM2_PCR_Allocate

23.5.1 General Description

This command is used to set the desired PCR allocation of PCR and algorithms. This command requires Platform Authorization.

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 this command is received multiple times before a TPM2_Startup(TPM_SU_CLEAR), each one overwrites the previous stored allocation.

This command will only change the allocations of banks that are listed in *pcrAllocation*.

EXAMPLE 1 If a TPM supports SHA1 and SHA256, then it maintains an allocation for two banks (one of which could be empty). If a TPM_PCR_ALLOCATE() only has a selector for the SHA1 bank, then only the allocation of the SHA1 bank will be changed and the SHA256 bank will remain unchanged. To change the allocation of a TPM from 24 SHA1 PCR and no SHA256 PCR to 24 SHA256 PCR and no SHA1 PCR, the pcrAllocation would have to have two selections: one for the empty SHA1 bank and one for the SHA256 bank with 24 PCR.

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.

When PCR are allocated, if DRTM_PCR is defined, the resulting allocation must have at least one bank with the DRTM PCR allocated. If HCRTM_PCR is defined, the resulting allocation must have at least one bank with the HCRTM_PCR allocated. If not, the TPM returns TPM_RC_PCR.

The TPM may return TPM_RC_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. Alternatively, if the request fails, The TPM may return TPM_RC_NO_RESULT.

EXAMPLE 2 An example for this type of failure is a TPM that can only support one bank at a time and cannot support arbitrary distribution of PCR among banks.

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).

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.5.2 Command and Response

Table 145 — TPM2_PCR_Allocate Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 146 — TPM2_PCR_Allocate Response

Type	Name	Description
TPM_ST	tag	see clause 7
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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.5.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PCR_Allocate_fp.h"
#ifdef TPM_CC_PCR_Allocate  // Conditional expansion of this file
```

Table 147 — TPM2_PCR_Allocate Errors

Error Returns	Meaning
TPM_RC_PCR	the allocation did not have required PCR
TPM_RC_NV_UNAVAILABLE	NV is not accessible
TPM_RC_NV_RATE	NV is in a rate-limiting mode

```
TPM_RC
TPM2_PCR_Allocate(
    PCR_Allocate_In     *in,            // IN: input parameter list
    PCR_Allocate_Out    *out            // OUT: output parameter list
    )
{
    TPM_RC      result;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point.
    // Note: These codes are not listed in the return values above because it is
    // an implementation choice to check in this routine rather than in a common
    // function that is called before these actions are called. These return values
    // are specified in the Response Code clause of this part of ISO/IEC 11889.
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS)
        return result;

// Command Output

    // Call PCR Allocation function.
    result = PCRAllocate(&in->pcrAllocation, &out->maxPCR,
                                        &out->sizeNeeded, &out->sizeAvailable);
    if(result == TPM_RC_PCR)
        return result;

    //
    out->allocationSuccess = (result == TPM_RC_SUCCESS);

    // if re-configuration succeeds, set the flag to indicate PCR configuration is
    // going to be changed in next boot
    if(out->allocationSuccess == YES)
        g_pcrReConfig = TRUE;

    return TPM_RC_SUCCESS;
}
#endif // CC_PCR_Allocate
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.6 TPM2_PCR_SetAuthPolicy

23.6.1 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 needs 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

23.6.2 Command and Response

Table 148 — TPM2_PCR_SetAuthPolicy Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 *authPolicy*
TPMI_ALG_HASH+	hashAlg	the hash algorithm of the policy
TPMI_DH_PCR	pcrNum	the PCR for which the policy is to be set

Table 149 — TPM2_PCR_SetAuthPolicy Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.6.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PCR_SetAuthPolicy_fp.h"
#ifdef TPM_CC_PCR_SetAuthPolicy  // Conditional expansion of this file
```

Table 150 — TPM2_PCR_SetAuthPolicy Errors

Error Returns	Meaning
TPM_RC_SIZE	size of *authPolicy* is not the size of a digest produced by *policyDigest*
TPM_RC_VALUE	PCR referenced by *pcrNum* is not a member of a PCR policy group

```
TPM_RC
TPM2_PCR_SetAuthPolicy(
    PCR_SetAuthPolicy_In    *in             // IN: input parameter list
    )
{
    UINT32      groupIndex;

    TPM_RC      result;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

// Input Validation:

    // Check the authPolicy consistent with hash algorithm
    if(in->authPolicy.t.size != CryptGetHashDigestSize(in->hashAlg))
        return TPM_RC_SIZE + RC_PCR_SetAuthPolicy_authPolicy;

    // If PCR does not belong to a policy group, return TPM_RC_VALUE
    if(!PCRBelongsPolicyGroup(in->pcrNum, &groupIndex))
        return TPM_RC_VALUE + RC_PCR_SetAuthPolicy_pcrNum;

// Internal Data Update

    // Set PCR policy
    gp.pcrPolicies.hashAlg[groupIndex] = in->hashAlg;
    gp.pcrPolicies.policy[groupIndex] = in->authPolicy;

    // Save new policy to NV
    NvWriteReserved(NV_PCR_POLICIES, &gp.pcrPolicies);

    return TPM_RC_SUCCESS;
}
#endif // CC_PCR_SetAuthPolicy
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.7 TPM2_PCR_SetAuthValue

23.7.1 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).

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.7.2 Command and Response

Table 151 — TPM2_PCR_SetAuthValue Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 152 — TPM2_PCR_SetAuthValue Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.7.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PCR_SetAuthValue_fp.h"
#ifdef TPM_CC_PCR_SetAuthValue  // Conditional expansion of this file
```

Table 153 — TPM2_PCR_SetAuthValue Errors

Error Returns	Meaning
TPM_RC_VALUE	PCR referenced by *pcrHandle* is not a member of a PCR authorization group

```
TPM_RC
TPM2_PCR_SetAuthValue(
    PCR_SetAuthValue_In     *in             // IN: input parameter list
    )
{
    UINT32      groupIndex;
    TPM_RC      result;

// Input Validation:

    // If PCR does not belong to an auth group, return TPM_RC_VALUE
    if(!PCRBelongsAuthGroup(in->pcrHandle, &groupIndex))
        return TPM_RC_VALUE;

    // The command may cause the orderlyState to be cleared due to the update of
    // state clear data.  If this is the case, Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    if(gp.orderlyState != SHUTDOWN_NONE)
    {
        result = NvIsAvailable();
        if(result != TPM_RC_SUCCESS) return result;
        g_clearOrderly = TRUE;
    }

// Internal Data Update

    // Set PCR authValue
    gc.pcrAuthValues.auth[groupIndex] = in->auth;

    return TPM_RC_SUCCESS;
}
#endif // CC_PCR_SetAuthValue
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.8 TPM2_PCR_Reset

23.8.1 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 (see ISO/IEC 11889-2, clause 10.6, “TPMI_DH_PCR”) 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

23.8.2 Command and Response

Table 154 — TPM2_PCR_Reset Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 155 — TPM2_PCR_Reset Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.8.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PCR_Reset_fp.h"
#ifdef TPM_CC_PCR_Reset  // Conditional expansion of this file
```

Table 156 — TPM2_PCR_Reset Errors

Error Returns	Meaning
TPM_RC_LOCALITY	current command locality is not allowed to reset the PCR referenced by *pcrHandle*

```
TPM_RC
TPM2_PCR_Reset(
    PCR_Reset_In    *in             // IN: input parameter list
    )
{
    TPM_RC      result;

// Input Validation

    // Check if the reset operation is allowed by the current command locality
    if(!PCRIsResetAllowed(in->pcrHandle))
        return TPM_RC_LOCALITY;

    // If PCR is state saved and we need to update orderlyState, check NV
    // availability
    if(PCRIsStateSaved(in->pcrHandle) && gp.orderlyState != SHUTDOWN_NONE)
    {
        result = NvIsAvailable();
        if(result != TPM_RC_SUCCESS)
            return result;
        g_clearOrderly = TRUE;
    }

// Internal Data Update

    // Reset selected PCR in all banks to 0
    PCRSetValue(in->pcrHandle, 0);

    // Indicate that the PCR changed so that pcrCounter will be incremented if
    // necessary.
    PCRChanged(in->pcrHandle);

    return TPM_RC_SUCCESS;
}
#endif // CC_PCR_Reset
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.9 _TPM_Hash_Start

23.9.1 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.9.2 Detailed Actions

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

This function is called to process a _TPM_Hash_Start() indication.

```
void
_TPM_Hash_Start(
    void
    )
{
    TPM_RC              result;
    TPMI_DH_OBJECT      handle;

    // If a DRTM sequence object exists, free it up
    if(g_DRTMHandle != TPM_RH_UNASSIGNED)
    {
        ObjectFlush(g_DRTMHandle);
        g_DRTMHandle = TPM_RH_UNASSIGNED;
    }

    // Create an event sequence object and store the handle in global
    // g_DRTMHandle. A TPM_RC_OBJECT_MEMORY error may be returned at this point
    // The null value for the 'auth' parameter will cause the sequence structure to
    // be allocated without being set as present. This keeps the sequence from
    // being left behind if the sequence is terminated early.
    result = ObjectCreateEventSequence(NULL, &g_DRTMHandle);

    // If a free slot was not available, then free up a slot.
    if(result != TPM_RC_SUCCESS)
    {
        // An implementation does not need to have a fixed relationship between
        // slot numbers and handle numbers. To handle the general case, scan for
        // a handle that is assigned and free it for the DRTM sequence.
        // In the reference implementation, the relationship between handles and
        // slots is fixed. So, if the call to ObjectCreateEvenSequence()
        // failed indicating that all slots are occupied, then the first handle we
        // are going to check (TRANSIENT_FIRST) will be occupied. It will be freed
        // so that it can be assigned for use as the DRTM sequence object.
        for(handle = TRANSIENT_FIRST; handle < TRANSIENT_LAST; handle++)
        {
            // try to flush the first object
            if(ObjectIsPresent(handle))
                break;
        }
        // If the first call to find a slot fails but none of the slots is occupied
        // then there's a big problem
        pAssert(handle < TRANSIENT_LAST);

        // Free the slot
        ObjectFlush(handle);

        // Try to create an event sequence object again.  This time, we must
        // succeed.
        result = ObjectCreateEventSequence(NULL, &g_DRTMHandle);
        pAssert(result == TPM_RC_SUCCESS);
    }

    return;
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.10 _TPM_Hash_Data

23.10.1 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 H-CRTM Event Sequence context exists, this indication is discarded and no other action is performed.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.10.2 Detailed Actions

```
#include "InternalRoutines.h"
#include "Platform.h"
#include "PCR_fp.h"
```

This function is called to process a _TPM_Hash_Data() indication.

```
void
_TPM_Hash_Data(
    UINT32           dataSize,      // IN: size of data to be extend
    BYTE            *data           // IN: data buffer
    )
{
    UINT32           i;
    HASH_OBJECT     *hashObject;
    TPMI_DH_PCR      pcrHandle = TPMIsStarted()
                                 ? PCR_FIRST + DRTM_PCR : PCR_FIRST + HCRTM_PCR;

    // If there is no DRTM sequence object, then _TPM_Hash_Start
    // was not called so this function returns without doing
    // anything.
    if(g_DRTMHandle == TPM_RH_UNASSIGNED)
        return;

    hashObject = (HASH_OBJECT *)ObjectGet(g_DRTMHandle);
    pAssert(hashObject->attributes.eventSeq);

    // For each of the implemented hash algorithms, update the digest with the
    // data provided.
    for(i = 0; i < HASH_COUNT; i++)
    {
        // make sure that the PCR is implemented for this algorithm
        if(PcrIsAllocated(pcrHandle,
                          hashObject->state.hashState[i].state.hashAlg))
            // Update sequence object
            CryptUpdateDigest(&hashObject->state.hashState[i], dataSize, data);
    }

    return;
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.11 _TPM_Hash_End

23.11.1 Description

This indication from the TPM interface indicates the end of the H-CRTM measurement. This indication is discarded and no other action performed if the TPM does not contain a H-CRTM Event Sequence context.

NOTE 1 An H-CRTM Event Sequence context is created by _TPM_Hash_Start().

If the H-CRTM Event Sequence occurs after TPM2_Startup(), the TPM will set all of the PCR designated in the platform-specific specifications as resettable 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 D-RTM PCR (PCR[17]).

$$\text{PCR}[17][hashAlg] := \mathbf{H}_{hashAlg}(initial_value \,||\, \mathbf{H}_{hashAlg}(hash_data)) \quad (7)$$

where

hashAlg	hash algorithm associated with a bank of PCR
initial_value	initialization value specified in the platform-specific specification (should be 0…0)
hash_data	all the octets of data received in _TPM_Hash_Data 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].

$$\text{PCR}[0][hashAlg] := \mathbf{H}_{hashAlg}(0...04 \,||\, \mathbf{H}_{hashAlg}(hash_data)) \quad (8)$$

NOTE 2 The entire sequence of _TPM_Hash_Start, _TPM_Hash_Data, and _TPM_Hash_End need to complete before TPM2_Startup() or the sequence will have no effect on the TPM.

NOTE 3 PCR[0] does not need to be updated according to (8) until the end of TPM2_Startup().

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

23.11.2 Detailed Actions

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

This function is called to process a _TPM_Hash_End() indication.

```
void
_TPM_Hash_End(
    void
    )
{

    UINT32          i;
    TPM2B_DIGEST    digest;
    HASH_OBJECT    *hashObject;
    TPMI_DH_PCR     pcrHandle;

    // If the DRTM handle is not being used, then either _TPM_Hash_Start has not
    // been called, _TPM_Hash_End was previously called, or some other command
    // was executed and the sequence was aborted.
    if(g_DRTMHandle == TPM_RH_UNASSIGNED)
        return;

    // Get DRTM sequence object
    hashObject = (HASH_OBJECT *)ObjectGet(g_DRTMHandle);


    // Is this _TPM_Hash_End after Startup or before
    if(TPMIsStarted())
    {
        // After

        // Reset the DRTM PCR
        PCRResetDynamics();

        // Extend the DRTM_PCR.
        pcrHandle = PCR_FIRST + DRTM_PCR;

        // DRTM sequence increments restartCount
        gr.restartCount++;
    }
    else
    {
        pcrHandle = PCR_FIRST + HCRTM_PCR;
    }


    // Complete hash and extend PCR, or if this is an HCRTM, complete
    // the hash, reset the H-CRTM register (PCR[0]) to 0...04, and then
    // extend the H-CRTM data
    for(i = 0; i < HASH_COUNT; i++)
    {
        TPMI_ALG_HASH       hash = CryptGetHashAlgByIndex(i);
        // make sure that the PCR is implemented for this algorithm
        if(PcrIsAllocated(pcrHandle,
                          hashObject->state.hashState[i].state.hashAlg))
        {
            // Complete hash
            digest.t.size = CryptGetHashDigestSize(hash);
            CryptCompleteHash2B(&hashObject->state.hashState[i], &digest.b);

            PcrDrtm(pcrHandle, hash, &digest);
        }
    }
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
60
61          // Flush sequence object.
62          ObjectFlush(g_DRTMHandle);
63
64          g_DRTMHandle = TPM_RH_UNASSIGNED;
65
66          g_DrtmPreStartup = TRUE;
67
68          return;
69      }
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24 Enhanced Authorization (EA) Commands

24.1 Introduction

The commands in clause 24 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 clause 24 are specified in detail in ISO/IEC 11889-1 and are not redefined in clause 24.

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 set to 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* can be updated for a trial policy but it is not required.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

24.2 Signed Authorization Actions

24.2.1 Introduction

The TPM2_PolicySigned, TPM_PolicySecret, and TPM2_PolicyTicket commands use many of the same functions. Clause 24.2 consolidates those functions to simplify this part of ISO/IEC 11889 and to ensure uniformity of the operations.

24.2.2 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. This parameter is required to be present if expiration is non-zero (TPM_RC_EXPIRED).

b) *expiration* – If this parameter is not zero, then its absolute value 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. If *nonceTPM* is the Empty buffer, and *expiration* is non-zero, then the TPM shall return TPM_RC_EXPIRED.

   If *policySession→timeout* is greater than *policySession→startTime* plus the absolute value of *expiration,* then *policySession→timeout* is set to *policySession→startTime* plus the absolute value of expiration. That is, *policySession→timeout* can only be changed to a smaller value.

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 needs to be the algorithm of the policy session.

   1) the TPM shall return TPM_RC_CPHASH if *policySession→cpHash* is set and 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*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.2.3 Policy Digest Update Function (PolicyUpdate())

This is the update process for *policySession→policyDigest* used by TPM2_PolicySigned(), TPM2_PolicySecret(), TPM2_PolicyTicket(), and TPM2_PolicyAuthorize(). The function prototype for the update function is:

$$\mathbf{PolicyUpdate}(commandCode, arg2, arg3) \tag{9}$$

where

arg2	a TPM2B_NAME
arg3	a TPM2B

These parameters are used to update *policySession→policyDigest* by

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, commandCode \,||\, arg2.name) \tag{10}$$

followed by

$$policyDigest_{new+1} := \mathbf{H}_{\text{policyAlg}}(policyDigest_{new} \,||\, arg3.buffer) \tag{11}$$

where

$\mathbf{H}_{policyAlg}()$	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 hash operations 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. For example, arg2 = 1 2 3 and arg3 = 4 5 6 would produce the same digest as arg2 = 1 2 and arg3 = 3 4 5 6. 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

24.2.4 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 changed 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.2.5 Policy Ticket Creation

If, for TPM2_PolicySigned() or TPM2_PolicySecret(), the caller specified a negative value for *expiration*, and the nonceTPM matches *policySession->nonceTPM*, then the TPM will return a ticket that includes a value indicating when the authorization expires. If *expiration* is non-negative, then the TPM will return a NULL ticket.

The required computation for the digest in the authorization ticket is:

$$\mathbf{HMAC}(proof, \mathbf{H}_{policyAlg}(ticketType \,||\, timeout \,||\, cpHashA \,||\, policyRef \,||\, authObject{\rightarrow}Name)) \quad (12)$$

where

proof	secret associated with the storage primary seed (SPS) of the TPM
$\boldsymbol{H}_{policyAlg}$	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; required to be the implementation equivalent of *policySession→startTime* plus the absolute value of *expiration*

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

24.3 TPM2_PolicySigned

24.3.1 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 specified in 24.2.3 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 entity will sign a digest of the authorization qualifiers: *nonceTPM*, *expiration*, *cpHashA*, and *policyRef*. The digest is computed as:

$$aHash := \mathbf{H}_{authAlg}(nonceTPM \,||\, expiration \,||\, cpHashA \,||\, policyRef) \quad (13)$$

where

$\mathbf{H}_{authAlg}()$ — the hash associated with the auth parameter of this command

1 Each signature and key combination indicates the scheme and each scheme has an NOTE 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.

NOTE 2 This parameter needs to be present if *expiration* is non-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 3 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:

$$aHash := \mathbf{H}_{authAlg}(00\ 00\ 00\ 00_{16})$$

which is the hash of an expiration time of zero.

The *aHash* is signed by the key associated with a key whose handle is *authObject*. The signature and signing parameters are combined to create the *auth* parameter.

The TPM will perform the parameter checks listed in 24.2.2

If the parameter checks succeed, the TPM will construct a test digest (*tHash*) over the provided parameters using the same formulation as 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*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

When all validations have succeeded, *policySession→policyDigest* is updated by **PolicyUpdate**() (see 24.2.3).

PolicyUpdate(TPM_CC_PolicySigned, *authObject→Name*, *policyRef*) (14)

policySession is updated as specified in 24.2.4. The TPM will optionally produce a ticket as specified in 24.2.5.

Authorization to use *authObject* is not required.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

24.3.2 Command and Response

Table 157 — TPM2_PolicySigned Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit, encrypt, or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicySigned
TPMI_DH_OBJECT	authObject	handle for a 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 This can be the Empty Buffer.
TPM2B_DIGEST	cpHashA	digest of the command parameters to which this authorization is limited This is not the *cpHash* for this command but the *cpHash* 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.
INT32	expiration	time when authorization will expire, measured in seconds from the time that *nonceTPM* was generated If *expiration* is non-negative, a NULL Ticket is returned. See 24.2.5.
TPMT_SIGNATURE	auth	signed authorization (not optional)

Table 158 — TPM2_PolicySigned Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_TIMEOUT	timeout	implementation-specific time value, used to indicate to the TPM when the ticket expires
TPMT_TK_AUTH	policyTicket	produced if the command succeeds and *expiration* in the command was non-zero; this ticket will use the TPMT_ST_AUTH_SIGNED structure tag. See 24.2.5
NOTE	If *policyTicket* is a NULL Ticket, then *timeout* needs to be the Empty Buffer.	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.3.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Policy_spt_fp.h"
#include "PolicySigned_fp.h"
#ifdef TPM_CC_PolicySigned  // Conditional expansion of this file
```

Table 159 — TPM2_PolicySigned Errors

Error Returns	Meaning
TPM_RC_CPHASH	*cpHash* was previously set to a different value
TPM_RC_EXPIRED	*expiration* indicates a time in the past or *expiration* is non-zero but no *nonceTPM* is present
TPM_RC_HANDLE	*authObject* need to have sensitive portion loaded
TPM_RC_KEY	*authObject* is not a signing scheme
TPM_RC_NONCE	*nonceTPM* is not the nonce associated with the *policySession*
TPM_RC_SCHEME	the signing scheme of *auth* is not supported by the TPM
TPM_RC_SIGNATURE	the signature is not genuine
TPM_RC_SIZE	input *cpHash* has wrong size
TPM_RC_VALUE	input *policyID* or expiration does not match the internal data in policy session

```
TPM_RC
TPM2_PolicySigned(
    PolicySigned_In     *in,            // IN: input parameter list
    PolicySigned_Out    *out            // OUT: output parameter list
    )
{
    TPM_RC                  result = TPM_RC_SUCCESS;
    SESSION                *session;
    TPM2B_NAME              entityName;
    TPM2B_DIGEST            authHash;
    HASH_STATE              hashState;
    UINT32                  expiration = (in->expiration < 0)
                                       ? -(in->expiration) : in->expiration;
    UINT64                  authTimeout = 0;

// Input Validation

    // Set up local pointers
    session = SessionGet(in->policySession);    // the session structure

    // Only do input validation if this is not a trial policy session
    if(session->attributes.isTrialPolicy == CLEAR)
    {
        if(expiration != 0)
            authTimeout = expiration * 1000 + session->startTime;

        result = PolicyParameterChecks(session, authTimeout,
                                       &in->cpHashA, &in->nonceTPM,
                                       RC_PolicySigned_nonceTPM,
                                       RC_PolicySigned_cpHashA,
                                       RC_PolicySigned_expiration);
        if(result != TPM_RC_SUCCESS)
            return result;

        // Re-compute the digest being signed
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

```
        /*(See this part of ISO/IEC 11889)
        // The digest is computed as:
        //     aHash := hash ( nonceTPM | expiration | cpHashA | policyRef)
        //  where:
        //      hash()      the hash associated with the signed auth
        //      nonceTPM    the nonceTPM value 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     hash of the command parameters for the command being
        //                  approved using the hash algorithm of the PSAP session.
        //                  Set to NULLauth if the authorization is not limited
        //                  to a specific command.
        //      policyRef   hash of an opaque value determined by the authorizing
        //                  object.  Set to the NULLdigest if no hash is present.
        */
        // Start hash
        authHash.t.size = CryptStartHash(CryptGetSignHashAlg(&in->auth),
                                         &hashState);

        //  add nonceTPM
        CryptUpdateDigest2B(&hashState, &in->nonceTPM.b);

        //  add expiration
        CryptUpdateDigestInt(&hashState, sizeof(UINT32), (BYTE*) &in->expiration);

        //  add cpHashA
        CryptUpdateDigest2B(&hashState, &in->cpHashA.b);

        //  add policyRef
        CryptUpdateDigest2B(&hashState, &in->policyRef.b);

        //  Complete digest
        CryptCompleteHash2B(&hashState, &authHash.b);

        // Validate Signature.  A TPM_RC_SCHEME, TPM_RC_HANDLE or TPM_RC_SIGNATURE
        // error may be returned at this point
        result = CryptVerifySignature(in->authObject, &authHash, &in->auth);
        if(result != TPM_RC_SUCCESS)
            return RcSafeAddToResult(result, RC_PolicySigned_auth);
    }
// Internal Data Update
    // Need the Name of the signing entity
    entityName.t.size = EntityGetName(in->authObject, &entityName.t.name);

    // Update policy with input policyRef and name of auth key
    // These values are updated even if the session is a trial session
    PolicyContextUpdate(TPM_CC_PolicySigned, &entityName, &in->policyRef,
                        &in->cpHashA, authTimeout, session);

// Command Output

    // Create ticket and timeout buffer if in->expiration < 0 and this is not
    // a trial session.
    // NOTE: PolicyParameterChecks() makes sure that nonceTPM is present
    // when expiration is non-zero.
    if(   in->expiration < 0
       && session->attributes.isTrialPolicy == CLEAR
      )
    {
        // Generate timeout buffer.  The format of output timeout buffer is
        // TPM-specific.
        // Note: can't do a direct copy because the output buffer is a byte
        // array and it may not be aligned to accept a 64-bit value.  The method
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
106             // used has the side-effect of making the returned value a big-endian,
107             // 64-bit value that is byte aligned.
108             out->timeout.t.size = sizeof(UINT64);
109             UINT64_TO_BYTE_ARRAY(authTimeout, out->timeout.t.buffer);
110 
111             // Compute policy ticket
112             TicketComputeAuth(TPM_ST_AUTH_SIGNED, EntityGetHierarchy(in->authObject),
113                               authTimeout, &in->cpHashA, &in->policyRef, &entityName,
114                               &out->policyTicket);
115         }
116         else
117         {
118             // Generate a null ticket.
119             // timeout buffer is null
120             out->timeout.t.size = 0;
121 
122             // auth ticket is null
123             out->policyTicket.tag = TPM_ST_AUTH_SIGNED;
124             out->policyTicket.hierarchy = TPM_RH_NULL;
125             out->policyTicket.digest.t.size = 0;
126         }
127 
128         return TPM_RC_SUCCESS;
129     }
130     #endif // CC_PolicySigned
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.4 TPM2_PolicySecret

24.4.1 General Description

This command includes a secret-based authorization to a policy. The caller proves knowledge of the secret value using an authorization session using the *authValue* associated with authHandle. A password session, an HMAC session, or a policy session containing TPM2_PolicyAuthValue() or TPM2_PolicyPassword() will satisfy this requirement.

If a policy session is used and use of the authValue of authHandle is not required, the TPM will return TPM_RC_MODE.

The secret is the *authValue* of the entity whose handle is *authHandle,* which may be any TPM entity with a handle and an associated *authValue*. This includes the reserved handles, NV Indexes, and loaded objects.

EXAMPLE Examples of reserved handles are Platform, Storage, and Endorsement.

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 specified in 24.2.2.

nonceTPM must be present if *expiration* is non-zero.

When all validations have succeeded, *policySession→policyDigest* is updated by **PolicyUpdate**() (see 24.2.3).

PolicyUpdate(TPM_CC_PolicySecret, *authObject→Name, policyRef*) (15)

policySession is updated as specified in 24.2.4. The TPM will optionally produce a ticket as specified in 24.2.5.

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().

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.4.2 Command and Response

Table 160 — TPM2_PolicySecret Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	Tag	TPM_ST_SESSIONS
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 This can be the Empty Buffer.
TPM2B_DIGEST	cpHashA	digest of the command parameters to which this authorization is limited This not the *cpHash* for this command but the *cpHash* 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.
INT32	Expiration	time when authorization will expire, measured in seconds from the time that *nonceTPM* was generated If *expiration* is non-negative, a NULL Ticket is returned. See 24.2.5.

Table 161 — TPM2_PolicySecret Response

Type	Name	Description
TPM_ST	Tag	see clause 7
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 *expiration* in the command was non-zero. See 24.2.5

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.4.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicySecret_fp.h"
#ifdef TPM_CC_PolicySecret  // Conditional expansion of this file
#include "Policy_spt_fp.h"
```

Table 162 — TPM2_PolicySecret Errors

Error Returns	Meaning
TPM_RC_CPHASH	*cpHash* for policy was previously set to a value that is not the same as *cpHashA*
TPM_RC_EXPIRED	*expiration* indicates a time in the past
TPM_RC_NONCE	*nonceTPM* does not match the nonce associated with *policySession*
TPM_RC_SIZE	*cpHashA* is not the size of a digest for the hash associated with *policySession*
TPM_RC_VALUE	input *policyID* or expiration does not match the internal data in policy session

```
TPM_RC
TPM2_PolicySecret(
    PolicySecret_In     *in,            // IN: input parameter list
    PolicySecret_Out    *out            // OUT: output parameter list
    )
{
    TPM_RC                  result;
    SESSION                *session;
    TPM2B_NAME              entityName;
    UINT32                  expiration = (in->expiration < 0)
                                       ? -(in->expiration) : in->expiration;
    UINT64                  authTimeout = 0;

// Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    //Only do input validation if this is not a trial policy session
    if(session->attributes.isTrialPolicy == CLEAR)
    {

        if(expiration != 0)
            authTimeout = expiration * 1000 + session->startTime;

        result = PolicyParameterChecks(session, authTimeout,
                                       &in->cpHashA, &in->nonceTPM,
                                       RC_PolicySecret_nonceTPM,
                                       RC_PolicySecret_cpHashA,
                                       RC_PolicySecret_expiration);
        if(result != TPM_RC_SUCCESS)
            return result;
    }

// Internal Data Update
    // Need the name of the authorizing entity
    entityName.t.size = EntityGetName(in->authHandle, &entityName.t.name);

    // Update policy context with input policyRef and name of auth key
    // This value is computed even for trial sessions. Possibly update the cpHash
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
45      PolicyContextUpdate(TPM_CC_PolicySecret, &entityName, &in->policyRef,
46                          &in->cpHashA, authTimeout, session);
47
48  // Command Output
49
50      // Create ticket and timeout buffer if in->expiration < 0 and this is not
51      // a trial session.
52      // NOTE: PolicyParameterChecks() makes sure that nonceTPM is present
53      // when expiration is non-zero.
54      if(   in->expiration < 0
55         && session->attributes.isTrialPolicy == CLEAR
56        )
57      {
58          // Generate timeout buffer.  The format of output timeout buffer is
59          // TPM-specific.
60          // Note: can't do a direct copy because the output buffer is a byte
61          // array and it may not be aligned to accept a 64-bit value.  The method
62          // used has the side-effect of making the returned value a big-endian,
63          // 64-bit value that is byte aligned.
64          out->timeout.t.size = sizeof(UINT64);
65          UINT64_TO_BYTE_ARRAY(authTimeout, out->timeout.t.buffer);
66
67          // Compute policy ticket
68          TicketComputeAuth(TPM_ST_AUTH_SECRET, EntityGetHierarchy(in->authHandle),
69                            authTimeout, &in->cpHashA, &in->policyRef,
70                            &entityName, &out->policyTicket);
71      }
72      else
73      {
74          // timeout buffer is null
75          out->timeout.t.size = 0;
76
77          // auth ticket is null
78          out->policyTicket.tag = TPM_ST_AUTH_SECRET;
79          out->policyTicket.hierarchy = TPM_RH_NULL;
80          out->policyTicket.digest.t.size = 0;
81      }
82
83      return TPM_RC_SUCCESS;
84  }
85  #endif // CC_PolicySecret
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.5 TPM2_PolicyTicket

24.5.1 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 specified in 24.2.2.

If the checks succeed, the TPM uses the *timeout*, *cpHashA*, *policyRef*, and *authName* 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 24.2.3).

$$\textbf{PolicyUpdate}(commandCode, authName, policyRef) \quad (16)$$

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.

policySession is updated as specified in 24.2.4.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.5.2 Command and Response

Table 163 — TPM2_PolicyTicket Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
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 164 — TPM2_PolicyTicket Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.5.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyTicket_fp.h"
#ifdef TPM_CC_PolicyTicket  // Conditional expansion of this file
#include "Policy_spt_fp.h"
```

Table 165 — TPM2_PolicyTicket Errors

Error Returns	Meaning
TPM_RC_CPHASH	policy's *cpHash* was previously set to a different value
TPM_RC_EXPIRED	*timeout* value in the ticket is in the past and the ticket has expired
TPM_RC_SIZE	*timeout* or *cpHash* has invalid size for the
TPM_RC_TICKET	*ticket* is not valid

```
TPM_RC
TPM2_PolicyTicket(
    PolicyTicket_In     *in             // IN: input parameter list
    )
{
    TPM_RC                  result;
    SESSION                *session;
    UINT64                  timeout;
    TPMT_TK_AUTH            ticketToCompare;
    TPM_CC                  commandCode = TPM_CC_PolicySecret;

// Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // NOTE: A trial policy session is not allowed to use this command.
    // A ticket is used in place of a previously given authorization. Since
    // a trial policy doesn't actually authenticate, the validated
    // ticket is not necessary and, in place of using a ticket, one
    // should use the intended authorization for which the ticket
    // would be a substitute.
    if(session->attributes.isTrialPolicy)
        return TPM_RCS_ATTRIBUTES + RC_PolicyTicket_policySession;

    // Restore timeout data.  The format of timeout buffer is TPM-specific.
    // In this implementation, we simply copy the value of timeout to the
    // buffer.
    if(in->timeout.t.size != sizeof(UINT64))
        return TPM_RC_SIZE + RC_PolicyTicket_timeout;
    timeout = BYTE_ARRAY_TO_UINT64(in->timeout.t.buffer);

    // Do the normal checks on the cpHashA and timeout values
    result = PolicyParameterChecks(session, timeout,
                                   &in->cpHashA, NULL,
                                   0,                       // no bad nonce return
                                   RC_PolicyTicket_cpHashA,
                                   RC_PolicyTicket_timeout);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Validate Ticket
    // Re-generate policy ticket by input parameters
    TicketComputeAuth(in->ticket.tag, in->ticket.hierarchy, timeout, &in->cpHashA,
                      &in->policyRef, &in->authName, &ticketToCompare);
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
50
51      // Compare generated digest with input ticket digest
52      if(!Memory2BEqual(&in->ticket.digest.b, &ticketToCompare.digest.b))
53          return TPM_RC_TICKET + RC_PolicyTicket_ticket;
54
55  // Internal Data Update
56
57      // Is this ticket to take the place of a TPM2_PolicySigned() or
58      // a TPM2_PolicySecret()?
59      if(in->ticket.tag == TPM_ST_AUTH_SIGNED)
60          commandCode = TPM_CC_PolicySigned;
61      else if(in->ticket.tag == TPM_ST_AUTH_SECRET)
62          commandCode = TPM_CC_PolicySecret;
63      else
64          // There could only be two possible tag values.  Any other value should
65          // be caught by the ticket validation process.
66          pAssert(FALSE);
67
68      // Update policy context
69      PolicyContextUpdate(commandCode, &in->authName, &in->policyRef,
70                          &in->cpHashA, timeout, session);
71
72      return TPM_RC_SUCCESS;
73  }
74  #endif // CC_PolicyTicket
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.6 TPM2_PolicyOR

24.6.1 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*:

$$digests := pHashList.digests[1].buffer \;||\; \ldots \;||\; pHashList.digests[n].buffer \tag{17}$$

NOTE 1 The TPM will not return an error if the size of an entry is not the same as the size of the digest of the policy. However, that entry cannot match *policyDigest*.

b) Reset *policyDigest* to a Zero Digest.

c) Extend the command code and the hashes computed in step a) above:

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \;||\; \text{TPM_CC_PolicyOR} \;||\; digests) \tag{18}$$

NOTE 2 The computation in b) and c) above is equivalent to:

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(0\ldots0 \;||\; \text{TPM_CC_PolicyOR} \;||\; digests)$$

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 ought to not use more than eight values.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.6.2 Command and Response

Table 166 — TPM2_PolicyOR Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
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 167 — TPM2_PolicyOR Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.6.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyOR_fp.h"
#ifdef TPM_CC_PolicyOR  // Conditional expansion of this file
#include "Policy_spt_fp.h"
```

Table 168 — TPM2_PolicyOR Errors

Error Returns	Meaning
TPM_RC_VALUE	no digest in *pHashList* matched the current value of *policyDigest* for *policySession*

```
TPM_RC
TPM2_PolicyOR(
    PolicyOR_In     *in             // IN: input parameter list
    )
{
    SESSION     *session;
    UINT32       i;

// Input Validation and Update

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Compare and Update Internal Session policy if match
    for(i = 0; i < in->pHashList.count; i++)
    {
        if(   session->attributes.isTrialPolicy == SET
           || (Memory2BEqual(&session->u2.policyDigest.b,
                             &in->pHashList.digests[i].b))
          )
        {
            // Found a match
            HASH_STATE      hashState;
            TPM_CC          commandCode = TPM_CC_PolicyOR;

            // Start hash
            session->u2.policyDigest.t.size = CryptStartHash(session->authHashAlg,
                                                             &hashState);
            // Set policyDigest to 0 string and add it to hash
            MemorySet(session->u2.policyDigest.t.buffer, 0,
                      session->u2.policyDigest.t.size);
            CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

            // add command code
            CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

            // Add each of the hashes in the list
            for(i = 0; i < in->pHashList.count; i++)
            {
                // Extend policyDigest
                CryptUpdateDigest2B(&hashState, &in->pHashList.digests[i].b);
            }
            // Complete digest
            CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);

            return TPM_RC_SUCCESS;
        }
    }
    // None of the values in the list matched the current policyDigest
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
54          return TPM_RC_VALUE + RC_PolicyOR_pHashList;
55      }
56      #endif // CC_PolicyOR
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.7 TPM2_PolicyPCR

24.7.1 General Description

This command is used to cause conditional gating of a policy based on PCR. This command together with TPM2_PolicyOR() 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 ISO/IEC 11889-1, clause 17.5, "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:

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, \text{TPM_CC_PolicyPCR} \,||\, pcrs \,||\, digestTPM) \qquad (19)$$

where

pcrs	the pcrs parameter with bits corresponding to unimplemented PCR set to 0
digestTPM	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.

NOTE 2 Since the pcrUpdateCounter is updated if any PCR is extended (except those specified not to do so), this means that the command will fail even if a PCR not specified in the policy is updated. This is an optimization for the purposes of conserving internal TPM memory. This would be a rare occurrence. In addition, if this should occur, the policy could be reset using the TPM2_PolicyRestart command and rerun.

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:

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, \text{TPM_CC_PolicyPCR} \,||\, pcrs \,||\, pcrDigest) \quad (20)$$

In this computation, pcrs is the input parameter without modification.

NOTE 3 The pcrs parameter is expected to match the configuration of the TPM for which the policy is being computed which might not be the same as the TPM on which the trial policy is being computed.

NOTE 4 Although no PCR are checked in a trial policy session, *pcrDigest* is expected to correspond to some useful PCR values. It is legal, but pointless, to have the TPM aid in calculating a *policyDigest* corresponding to PCR values that are not useful in practice.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

24.7.2 Command and Response

Table 169 — TPM2_PolicyPCR Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyPCR
TPMI_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 170 — TPM2_PolicyPCR Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.7.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyPCR_fp.h"
#ifdef TPM_CC_PolicyPCR  // Conditional expansion of this file
```

Table 171 — TPM2_PolicyPCR Errors

Error Returns	Meaning
TPM_RC_VALUE	if provided, *pcrDigest* does not match the current PCR settings
TPM_RC_PCR_CHANGED	a previous TPM2_PolicyPCR() set *pcrCounter* and it has changed

```
TPM_RC
TPM2_PolicyPCR(
    PolicyPCR_In    *in             // IN: input parameter list
    )
{
    SESSION         *session;
    TPM2B_DIGEST     pcrDigest;
    BYTE             pcrs[sizeof(TPML_PCR_SELECTION)];
    UINT32           pcrSize;
    BYTE            *buffer;
    TPM_CC           commandCode = TPM_CC_PolicyPCR;
    HASH_STATE       hashState;

// Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Do validation for non trial session
    if(session->attributes.isTrialPolicy == CLEAR)
    {
        // Make sure that this is not going to invalidate a previous PCR check
        if(session->pcrCounter != 0 && session->pcrCounter != gr.pcrCounter)
            return TPM_RC_PCR_CHANGED;

        // Compute current PCR digest
        PCRComputeCurrentDigest(session->authHashAlg, &in->pcrs, &pcrDigest);

        // If the caller specified the PCR digest and it does not
        // match the current PCR settings, return an error..
        if(in->pcrDigest.t.size != 0)
        {
            if(!Memory2BEqual(&in->pcrDigest.b, &pcrDigest.b))
                return TPM_RC_VALUE + RC_PolicyPCR_pcrDigest;
        }
    }
    else
    {
        // For trial session, just use the input PCR digest
        pcrDigest = in->pcrDigest;
    }
// Internal Data Update

    // Update policy hash
    // policyDigestnew = hash(   policyDigestold || TPM_CC_PolicyPCR
    //                        || pcrs || pcrDigest)
    //  Start hash
    CryptStartHash(session->authHashAlg, &hashState);

    //  add old digest
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
54          CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
55
56          //  add commandCode
57          CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
58
59          //  add PCRS
60          buffer = pcrs;
61          pcrSize = TPML_PCR_SELECTION_Marshal(&in->pcrs, &buffer, NULL);
62          CryptUpdateDigest(&hashState, pcrSize, pcrs);
63
64          //  add PCR digest
65          CryptUpdateDigest2B(&hashState, &pcrDigest.b);
66
67          //  complete the hash and get the results
68          CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
69
70          //  update pcrCounter in session context for non trial session
71          if(session->attributes.isTrialPolicy == CLEAR)
72          {
73              session->pcrCounter = gr.pcrCounter;
74          }
75
76          return TPM_RC_SUCCESS;
77      }
78      #endif // CC_PolicyPCR
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.8 TPM2_PolicyLocality

24.8.1 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. When the policy session is started, this parameter is initialized to a value that allows 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 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

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, \text{TPM_CC_PolicyLocality} \,||\, locality) \quad (21)$$

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*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

24.8.2 Command and Response

Table 172 — TPM2_PolicyLocality Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
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 173 — TPM2_PolicyLocality Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.8.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyLocality_fp.h"
#ifdef TPM_CC_PolicyLocality  // Conditional expansion of this file
```

Limit a policy to a specific locality

Table 174 — TPM2_PolicyLocality Errors

Error Returns	Meaning
TPM_RC_RANGE	all the locality values selected by *locality* have been disabled by previous TPM2_PolicyLocality() calls.

```
TPM_RC
TPM2_PolicyLocality(
    PolicyLocality_In   *in             // IN: input parameter list
    )
{
    SESSION     *session;
    BYTE         marshalBuffer[sizeof(TPMA_LOCALITY)];
    BYTE         prevSetting[sizeof(TPMA_LOCALITY)];
    UINT32       marshalSize;
    BYTE        *buffer;
    TPM_CC       commandCode = TPM_CC_PolicyLocality;
    HASH_STATE   hashState;

// Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Get new locality setting in canonical form
    buffer = marshalBuffer;
    marshalSize = TPMA_LOCALITY_Marshal(&in->locality, &buffer, NULL);

    // Its an error if the locality parameter is zero
    if(marshalBuffer[0] == 0)
        return TPM_RC_RANGE + RC_PolicyLocality_locality;

    // Get existing locality setting in canonical form
    buffer = prevSetting;
    TPMA_LOCALITY_Marshal(&session->commandLocality, &buffer, NULL);

    // If the locality has previously been set
    if(    prevSetting[0] != 0
        // then the current locality setting and the requested have to be the same
        // type (that is, either both normal or both extended
        && ((prevSetting[0] < 32) != (marshalBuffer[0] < 32)))
        return TPM_RC_RANGE + RC_PolicyLocality_locality;

    // See if the input is a regular or extended locality
    if(marshalBuffer[0] < 32)
    {
        // if there was no previous setting, start with all normal localities
        // enabled
        if(prevSetting[0] == 0)
            prevSetting[0] = 0x1F;

        // AND the new setting with the previous setting and store it in prevSetting
        prevSetting[0] &= marshalBuffer[0];

```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
52          // The result setting can not be 0
53          if(prevSetting[0] == 0)
54              return TPM_RC_RANGE + RC_PolicyLocality_locality;
55      }
56      else
57      {
58          // for extended locality
59          // if the locality has already been set, then it must match the
60          if(prevSetting[0] != 0 && prevSetting[0] != marshalBuffer[0])
61              return TPM_RC_RANGE + RC_PolicyLocality_locality;
62
63          // Setting is OK
64          prevSetting[0] = marshalBuffer[0];
65
66      }
67
68  // Internal Data Update
69
70      // Update policy hash
71      // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyLocality || locality)
72      // Start hash
73      CryptStartHash(session->authHashAlg, &hashState);
74
75      // add old digest
76      CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
77
78      // add commandCode
79      CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
80
81      // add input locality
82      CryptUpdateDigest(&hashState, marshalSize, marshalBuffer);
83
84      // complete the digest
85      CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
86
87      // update session locality by unmarshal function.  The function must succeed
88      // because both input and existing locality setting have been validated.
89      buffer = prevSetting;
90      TPMA_LOCALITY_Unmarshal(&session->commandLocality, &buffer,
91                              (INT32 *) &marshalSize);
92
93      return TPM_RC_SUCCESS;
94  }
95  #endif // CC_PolicyLocality
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.9 TPM2_PolicyNV

24.9.1 General Description

This command is used to cause conditional gating of a policy based on the contents of an NV Index. It is an immediate assertion. The NV index is validated during the TPM2_PolicyNV() command, not when the session is used for authorization.

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 If read access is controlled by policy, the policy ought to 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.

operandA begins at *offest* into the NV index contents and has a size equal to the size of *operandB*. The TPM will perform the indicated arithmetic check using *operandA* and *operandB*. . 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:

$$args := \mathbf{H}_{policyAlg}(operandB.buffer \,||\, offset \,||\, operation) \quad (22)$$

where

$\mathbf{H}_{policyAlg}()$	hash function using the algorithm of the policy session
$operandB$	the value used for the comparison
$offset$	offset from the start of the NV Index data to start the comparison
operation	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

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, \text{TPM_CC_PolicyNV} \,||\, args \,||\, nvIndex{\rightarrow}Name) \quad (23)$$

where

$\mathbf{H}_{policyAlg}()$	hash function using the algorithm of the policy session
$args$	value computed in equation (22)
$nvIndex{\rightarrow}Name$	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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

24.9.2 Command and Response

Table 175 — TPM2_PolicyNV Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 176 — TPM2_PolicyNV Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.9.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyNV_fp.h"
#ifdef TPM_CC_PolicyNV  // Conditional expansion of this file
#include "Policy_spt_fp.h"
#include "NV_spt_fp.h"          // Include NV support routine for read access check
```

Table 177 — TPM2_PolicyNV Errors

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 *nvIndex* data starting at *offset* is less than the size of *operandB*

```
TPM_RC
TPM2_PolicyNV(
    PolicyNV_In     *in             // IN: input parameter list
    )
{
    TPM_RC              result;
    SESSION            *session;
    NV_INDEX            nvIndex;
        BYTE            nvBuffer[sizeof(in->operandB.t.buffer)];
    TPM2B_NAME          nvName;
    TPM_CC              commandCode = TPM_CC_PolicyNV;
    HASH_STATE          hashState;
    TPM2B_DIGEST        argHash;

// Input Validation

    // Get NV index information
    NvGetIndexInfo(in->nvIndex, &nvIndex);

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    //If this is a trial policy, skip all validations and the operation
    if(session->attributes.isTrialPolicy == CLEAR)
    {
        // NV Read access check.  NV index should be allowed for read.  A
        // TPM_RC_AUTH_TYPE or TPM_RC_NV_LOCKED error may be return at this
        // point
        result = NvReadAccessChecks(in->authHandle, in->nvIndex);
        if(result != TPM_RC_SUCCESS) return result;

        // Valid NV data size should not be smaller than input operandB size
        if((nvIndex.publicArea.dataSize - in->offset) < in->operandB.t.size)
            return TPM_RC_SIZE + RC_PolicyNV_operandB;

        // Arithmetic Comparison

        // Get NV data.  The size of NV data equals the input operand B size
        NvGetIndexData(in->nvIndex, &nvIndex, in->offset,
                       in->operandB.t.size, nvBuffer);

```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
47          switch(in->operation)
48          {
49              case TPM_EO_EQ:
50                  // compare A = B
51                  if(CryptCompare(in->operandB.t.size, nvBuffer,
52                                  in->operandB.t.size, in->operandB.t.buffer) != 0)
53                      return TPM_RC_POLICY;
54                  break;
55              case TPM_EO_NEQ:
56                  // compare A != B
57                  if(CryptCompare(in->operandB.t.size, nvBuffer,
58                                  in->operandB.t.size, in->operandB.t.buffer) == 0)
59                      return TPM_RC_POLICY;
60                  break;
61              case TPM_EO_SIGNED_GT:
62                  // compare A > B signed
63                  if(CryptCompareSigned(in->operandB.t.size, nvBuffer,
64                                  in->operandB.t.size, in->operandB.t.buffer) <= 0)
65                      return TPM_RC_POLICY;
66                  break;
67              case TPM_EO_UNSIGNED_GT:
68                  // compare A > B unsigned
69                  if(CryptCompare(in->operandB.t.size, nvBuffer,
70                                  in->operandB.t.size, in->operandB.t.buffer) <= 0)
71                      return TPM_RC_POLICY;
72                  break;
73              case TPM_EO_SIGNED_LT:
74                  // compare A < B signed
75                  if(CryptCompareSigned(in->operandB.t.size, nvBuffer,
76                                  in->operandB.t.size, in->operandB.t.buffer) >= 0)
77                      return TPM_RC_POLICY;
78                  break;
79              case TPM_EO_UNSIGNED_LT:
80                  // compare A < B unsigned
81                  if(CryptCompare(in->operandB.t.size, nvBuffer,
82                                  in->operandB.t.size, in->operandB.t.buffer) >= 0)
83                      return TPM_RC_POLICY;
84                  break;
85              case TPM_EO_SIGNED_GE:
86                  // compare A >= B signed
87                  if(CryptCompareSigned(in->operandB.t.size, nvBuffer,
88                                  in->operandB.t.size, in->operandB.t.buffer) < 0)
89                      return TPM_RC_POLICY;
90                  break;
91              case TPM_EO_UNSIGNED_GE:
92                  // compare A >= B unsigned
93                  if(CryptCompare(in->operandB.t.size, nvBuffer,
94                                  in->operandB.t.size, in->operandB.t.buffer) < 0)
95                      return TPM_RC_POLICY;
96                  break;
97              case TPM_EO_SIGNED_LE:
98                  // compare A <= B signed
99                  if(CryptCompareSigned(in->operandB.t.size, nvBuffer,
100                                 in->operandB.t.size, in->operandB.t.buffer) > 0)
101                     return TPM_RC_POLICY;
102                 break;
103             case TPM_EO_UNSIGNED_LE:
104                 // compare A <= B unsigned
105                 if(CryptCompare(in->operandB.t.size, nvBuffer,
106                                 in->operandB.t.size, in->operandB.t.buffer) > 0)
107                     return TPM_RC_POLICY;
108                 break;
109             case TPM_EO_BITSET:
110                 // All bits SET in B are SET in A. ((A&B)=B)
111                 {
112                     UINT32 i;
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
                for (i = 0; i < in->operandB.t.size; i++)
                    if((nvBuffer[i] & in->operandB.t.buffer[i])
                            != in->operandB.t.buffer[i])
                        return TPM_RC_POLICY;
            }
            break;
            case TPM_EO_BITCLEAR:
                // All bits SET in B are CLEAR in A. ((A&B)=0)
            {
                UINT32 i;
                for (i = 0; i < in->operandB.t.size; i++)
                    if((nvBuffer[i] & in->operandB.t.buffer[i]) != 0)
                        return TPM_RC_POLICY;
            }
            break;
            default:
                pAssert(FALSE);
                break;
        }
    }

// Internal Data Update

    // Start argument hash
    argHash.t.size = CryptStartHash(session->authHashAlg, &hashState);

    //  add operandB
    CryptUpdateDigest2B(&hashState, &in->operandB.b);

    //  add offset
    CryptUpdateDigestInt(&hashState, sizeof(UINT16), &in->offset);

    //  add operation
    CryptUpdateDigestInt(&hashState, sizeof(TPM_EO), &in->operation);

    //  complete argument digest
    CryptCompleteHash2B(&hashState, &argHash.b);

    // Update policyDigest
    //  Start digest
    CryptStartHash(session->authHashAlg, &hashState);

    //  add old digest
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

    //  add argument digest
    CryptUpdateDigest2B(&hashState, &argHash.b);

    // Adding nvName
    nvName.t.size = EntityGetName(in->nvIndex, &nvName.t.name);
    CryptUpdateDigest2B(&hashState, &nvName.b);

    // complete the digest
    CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);

    return TPM_RC_SUCCESS;
}
#endif // CC_PolicyNV
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.10 TPM2_PolicyCounterTimer

24.10.1 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:

$$args := \mathbf{H}_{policyAlg}(operandB.buffer \,||\, offset \,||\, operation) \quad (24)$$

where

$\mathbf{H}_{policyAlg}()$	hash function using the algorithm of the policy session
$operandB.buffer$	the value used for the comparison
$offset$	offset from the start of the TPMS_TIME_INFO structure at which the comparison starts
$operation$	the operation parameter indicating the comparison being performed

The value of *args* is extended to *policySession→policyDigest* by

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, \text{TPM_CC_PolicyCounterTimer} \,||\, args) \quad (25)$$

where

$\mathbf{H}_{policyAlg}()$	hash function using the algorithm of the policy session
$args$	value computed in equation (24)

The signed arithmetic operations are performed using twos-compliment. The indicated portion of the TPMS_TIME_INFO structure begins at *offset* and has a length of *operandB.size*. If the octets to be compared overflows the TPMS_TIME_INFO structure, the TPM returns TPM_RC_RANGE. The structure is marshaled into its canonical form with no padding. The TPM does not check for alignment of the offset with a TPMS_TIME_INFO structure member.

Magnitude comparisons assume that the octet at offset zero in the referenced location and in *operandB* contain the most significant octet of the data.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.10.2 Command and Response

Table 178 — TPM2_PolicyCounterTimer Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
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 179 — TPM2_PolicyCounterTimer Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.10.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyCounterTimer_fp.h"
#ifdef TPM_CC_PolicyCounterTimer  // Conditional expansion of this file
#include "Policy_spt_fp.h"
```

Table 180 — TPM2_PolicyCounterTimer Errors

Error Returns	Meaning
TPM_RC_POLICY	the comparison of the selected portion of the TPMS_TIME_INFO with *operandB* failed
TPM_RC_RANGE	*offset* + *size* exceed size of TPMS_TIME_INFO structure

```
TPM_RC
TPM2_PolicyCounterTimer(
    PolicyCounterTimer_In   *in             // IN: input parameter list
    )
{
    TPM_RC              result;
    SESSION             *session;
    TIME_INFO           infoData;      // data buffer of  TPMS_TIME_INFO
    TPM_CC              commandCode = TPM_CC_PolicyCounterTimer;
    HASH_STATE          hashState;
    TPM2B_DIGEST        argHash;

// Input Validation

    // If the command is going to use any part of the counter or timer, need
    // to verify that time is advancing.
    // The time and clock vales are the first two 64-bit values in the clock
    if(in->offset < <K>sizeof(UINT64) + sizeof(UINT64))
    {
        // Using Clock or Time so see if clock is running. Clock doesn't run while
        // NV is unavailable.
        // TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned here.
        result = NvIsAvailable();
        if(result != TPM_RC_SUCCESS)
            return result;
    }
    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    //If this is a trial policy, skip all validations and the operation
    if(session->attributes.isTrialPolicy == CLEAR)
    {
        // Get time data info.  The size of time info data equals the input
        // operand B size.  A TPM_RC_RANGE error may be returned at this point
        result = TimeGetRange(in->offset, in->operandB.t.size, &infoData);
        if(result != TPM_RC_SUCCESS) return result;

        // Arithmetic Comparison
        switch(in->operation)
        {
            case TPM_EO_EQ:
                // compare A = B
                if(CryptCompare(in->operandB.t.size, infoData,
                                in->operandB.t.size, in->operandB.t.buffer) != 0)
                    return TPM_RC_POLICY;
                break;
            case TPM_EO_NEQ:
                // compare A != B
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
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;
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

```
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
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 }
161 #endif // CC_PolicyCounterTimer
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.11 TPM2_PolicyCommandCode

24.11.1 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

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, \text{TPM_CC_PolicyCommandCode} \,||\, code) \quad (26)$$

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 necessary.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

24.11.2 Command and Response

Table 181 — TPM2_PolicyCommandCode Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
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 *commandCode*

Table 182 — TPM2_PolicyCommandCode Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.11.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyCommandCode_fp.h"
#ifdef TPM_CC_PolicyCommandCode  // Conditional expansion of this file
```

Table 183 — TPM2_PolicyCommandCode Errors

Error Returns	Meaning
TPM_RC_VALUE	*commandCode* of *policySession* previously set to a different value

```
TPM_RC
TPM2_PolicyCommandCode(
    PolicyCommandCode_In    *in             // IN: input parameter list
    )
{
    SESSION     *session;
    TPM_CC      commandCode = TPM_CC_PolicyCommandCode;
    HASH_STATE  hashState;

// Input validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    if(session->commandCode != 0 && session->commandCode != in->code)
        return TPM_RC_VALUE + RC_PolicyCommandCode_code;
    if(!CommandIsImplemented(in->code))
        return TPM_RC_POLICY_CC + RC_PolicyCommandCode_code;

// Internal Data Update
    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyCommandCode || code)
    //  Start hash
    CryptStartHash(session->authHashAlg, &hashState);

    //  add old digest
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

    //  add input commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &in->code);

    //  complete the hash and get the results
    CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);

    // update commandCode value in session context
    session->commandCode = in->code;

    return TPM_RC_SUCCESS;
}
#endif // CC_PolicyCommandCode
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.12 TPM2_PolicyPhysicalPresence

24.12.1 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

$$policyDigest_{new} \coloneqq \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, \text{TPM_CC_PolicyPhysicalPresence}) \quad (27)$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.12.2 Command and Response

Table 184 — TPM2_PolicyPhysicalPresence Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyPhysicalPresence
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None

Table 185 — TPM2_PolicyPhysicalPresence Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.12.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyPhysicalPresence_fp.h"
#ifdef TPM_CC_PolicyPhysicalPresence  // Conditional expansion of this file
TPM_RC
TPM2_PolicyPhysicalPresence(
    PolicyPhysicalPresence_In   *in             // IN: input parameter list
    )
{
    SESSION     *session;
    TPM_CC      commandCode = TPM_CC_PolicyPhysicalPresence;
    HASH_STATE  hashState;

// Internal Data Update

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyPhysicalPresence)
    //  Start hash
    CryptStartHash(session->authHashAlg, &hashState);

    //  add old digest
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

    //  complete the digest
    CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);

    // update session attribute
    session->attributes.isPPRequired = SET;

    return TPM_RC_SUCCESS;
}
#endif // CC_PolicyPhysicalPresence
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.13 TPM2_PolicyCpHash

24.13.1 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

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, \text{TPM_CC_PolicyCpHash} \,||\, cpHashA) \quad (28)$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.13.2 Command and Response

Table 186 — TPM2_PolicyCpHash Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
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 *cpHash* added to the policy

Table 187 — TPM2_PolicyCpHash Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.13.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyCpHash_fp.h"
#ifdef TPM_CC_PolicyCpHash  // Conditional expansion of this file
```

Table 188 — TPM2_PolicyCpHash Errors

Error Returns	Meaning
TPM_RC_CPHASH	*cpHash* of *policySession* has previously been set to a different value
TPM_RC_SIZE	*cpHashA* is not the size of a digest produced by the hash algorithm associated with *policySession*

```
TPM_RC
TPM2_PolicyCpHash(
    PolicyCpHash_In     *in             // IN: input parameter list
    )
{
    SESSION     *session;
    TPM_CC      commandCode = TPM_CC_PolicyCpHash;
    HASH_STATE  hashState;

// Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // A new cpHash is given in input parameter, but cpHash in session context
    // is not empty, or is not the same as the new cpHash
    if(    in->cpHashA.t.size != 0
        && session->u1.cpHash.t.size != 0
        && !Memory2BEqual(&in->cpHashA.b, &session->u1.cpHash.b)
       )
        return TPM_RC_CPHASH;

    // A valid cpHash must have the same size as session hash digest
    if(in->cpHashA.t.size != CryptGetHashDigestSize(session->authHashAlg))
        return TPM_RC_SIZE + RC_PolicyCpHash_cpHashA;

// Internal Data Update

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyCpHash || cpHashA)
    //  Start hash
    CryptStartHash(session->authHashAlg, &hashState);

    //  add old digest
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

    //  add cpHashA
    CryptUpdateDigest2B(&hashState, &in->cpHashA.b);

    //  complete the digest and get the results
    CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);

    // update cpHash in session context
    session->u1.cpHash = in->cpHashA;
    session->attributes.iscpHashDefined = SET;

```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
53          return TPM_RC_SUCCESS;
54      }
55      #endif // CC_PolicyCpHash
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.14 TPM2_PolicyNameHash

24.14.1 General Description

This command allows a policy to be bound to a specific set of TPM entities without being bound to the parameters of the command. This is most useful for commands when the referenced PCR requires a policy.

EXAMPLE 1 Examples of commands when the referenced PCR requires a policy are TPM2_Duplicate() and TPM2_PCR_Event().

The *nameHash* parameter should contain the digest of the Names associated with the handles to be used in the authorized command.

EXAMPLE 2 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:

$$nameHash := \mathbf{H}_{policyAlg}(objectHandle{\rightarrow}Name \,||\, newParentHandle{\rightarrow}Name)$$

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

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, \text{TPM_CC_PolicyNameHash} \,||\, nameHash) \quad (29)$$

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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

24.14.2 Command and Response

Table 189 — TPM2_PolicyNameHash Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
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 190 — TPM2_PolicyNameHash Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.14.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyNameHash_fp.h"
#ifdef TPM_CC_PolicyNameHash  // Conditional expansion of this file
```

Table 191 — TPM2_PolicyNameHash Errors

Error Returns	Meaning
TPM_RC_CPHASH	*nameHash* has been previously set to a different value
TPM_RC_SIZE	*nameHash* is not the size of the digest produced by the hash algorithm associated with *policySession*

```
TPM_RC
TPM2_PolicyNameHash(
    PolicyNameHash_In   *in             // IN: input parameter list
    )
{
    SESSION             *session;
    TPM_CC               commandCode = TPM_CC_PolicyNameHash;
    HASH_STATE           hashState;

// Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // A new nameHash is given in input parameter, but cpHash in session context
    // is not empty
    if(in->nameHash.t.size != 0 && session->u1.cpHash.t.size != 0)
        return TPM_RC_CPHASH;

    // A valid nameHash must have the same size as session hash digest
    if(in->nameHash.t.size != CryptGetHashDigestSize(session->authHashAlg))
        return TPM_RC_SIZE + RC_PolicyNameHash_nameHash;

// Internal Data Update

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyNameHash || nameHash)
    //  Start hash
    CryptStartHash(session->authHashAlg, &hashState);

    //  add old digest
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

    //  add nameHash
    CryptUpdateDigest2B(&hashState, &in->nameHash.b);

    //  complete the digest
    CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);

    //  clear iscpHashDefined bit to indicate now this field contains a nameHash
    session->attributes.iscpHashDefined = CLEAR;

    // update nameHash in session context
    session->u1.cpHash = in->nameHash;

    return TPM_RC_SUCCESS;
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
53    }
54    #endif // CC_PolicyNameHash
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.15 TPM2_PolicyDuplicationSelect

24.15.1 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 can 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 can 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→nameHash* will be set to:

$$nameHash := \mathbf{H}_{policyAlg}(objectName \,||\, newParentName) \tag{30}$$

NOTE 3 It is allowed that policySesion→nameHash and policySession→cpHash 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:

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, \text{TPM_CC_PolicyDuplicationSelect} \,||\, objectName \,||\, newParentName \,||\, includeObject) \tag{31}$$

If includeObject is NO, *policySession→policyDigest* is updated by:

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, \text{TPM_CC_PolicyDuplicationSelect} \,||\, newParentName \,||\, includeObject) \tag{32}$$

NOTE 4 *policySession→cpHash* receives the digest of both Names so that the check performed in TPM2_Duplicate() can 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, *policySession→commandCode* 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 might 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).

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

24.15.2 Command and Response

Table 192 — TPM2_PolicyDuplicationSelect Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
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 *objectName* will be included in the value in *policySession→policyDigest*

Table 193 — TPM2_PolicyDuplicationSelect Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.15.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyDuplicationSelect_fp.h"
#ifdef TPM_CC_PolicyDuplicationSelect  // Conditional expansion of this file
```

Table 194 — TPM2_PolicyDuplicationSelect Errors

Error Returns	Meaning
TPM_RC_COMMAND_CODE	*commandCode* of '*policySession*; is not empty
TPM_RC_CPHASH	*cpHash* of *policySession* is not empty

```
TPM_RC
TPM2_PolicyDuplicationSelect(
    PolicyDuplicationSelect_In  *in             // IN: input parameter list
    )
{
    SESSION         *session;
    HASH_STATE      hashState;
    TPM_CC          commandCode = TPM_CC_PolicyDuplicationSelect;

// Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // cpHash in session context must be empty
    if(session->u1.cpHash.t.size != 0)
        return TPM_RC_CPHASH;

    // commandCode in session context must be empty
    if(session->commandCode != 0)
        return TPM_RC_COMMAND_CODE;

// Internal Data Update

    // Update name hash
    session->u1.cpHash.t.size = CryptStartHash(session->authHashAlg, &hashState);

    //  add objectName
    CryptUpdateDigest2B(&hashState, &in->objectName.b);

    //  add new parent name
    CryptUpdateDigest2B(&hashState, &in->newParentName.b);

    //  complete hash
    CryptCompleteHash2B(&hashState, &session->u1.cpHash.b);

    // update policy hash
    // Old policyDigest size should be the same as the new policyDigest size since
    // they are using the same hash algorithm
    session->u2.policyDigest.t.size
            = CryptStartHash(session->authHashAlg, &hashState);

    //  add old policy
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

    //  add command code
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

    //  add objectName
    if(in->includeObject == YES)
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
54              CryptUpdateDigest2B(&hashState, &in->objectName.b);
55
56          //  add new parent name
57          CryptUpdateDigest2B(&hashState, &in->newParentName.b);
58
59          //  add includeObject
60          CryptUpdateDigestInt(&hashState, sizeof(TPMI_YES_NO), &in->includeObject);
61
62          //  complete digest
63          CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
64
65          // clear iscpHashDefined bit to indicate now this field contains a nameHash
66          session->attributes.iscpHashDefined = CLEAR;
67
68          // set commandCode in session context
69          session->commandCode = TPM_CC_Duplicate;
70
71          return TPM_RC_SUCCESS;
72      }
73      #endif // CC_PolicyDuplicationSelect
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.16 TPM2_PolicyAuthorize

24.16.1 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

$$aHash := \mathbf{H}_{aHashAlg}(approvedPolicy \,||\, policyRef) \tag{33}$$

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

$$\mathbf{HMAC}(proof, (\text{TPM_ST_VERIFIED} \,||\, aHash \,||\, keySign{\rightarrow}Name)) \tag{34}$$

NOTE 1 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 update *policySession→policyDigest* with **PolicyUpdate**() (see 24.2.3).

$$\mathbf{PolicyUpdate}(\text{TPM_CC_PolicyAuthorize}, keySign, policyRef) \tag{35}$$

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 2 The unmarshaling process requires that a proper TPMT_TK_VERIFIED be provided for *checkTicket* but it can be a NULL Ticket. A NULL ticket is useful in a trial policy, where the caller uses the TPM to perform policy calculations but does not have a valid authorization ticket.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

24.16.2 Command and Response

Table 195 — TPM2_PolicyAuthorize Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
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 *approvedPolicy* and *policyRef* were signed by *keySign*

Table 196 — TPM2_PolicyAuthorize Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.16.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyAuthorize_fp.h"
#ifdef TPM_CC_PolicyAuthorize  // Conditional expansion of this file
#include "Policy_spt_fp.h"
```

Table 197 — TPM2_PolicyAuthorize Errors

Error Returns	Meaning
TPM_RC_HASH	hash algorithm in *keyName* is not supported
TPM_RC_SIZE	*keyName* is not the correct size for its hash algorithm
TPM_RC_VALUE	the current *policyDigest* of *policySession* does not match *approvedPolicy*; or *checkTicket* doesn't match the provided values

```
TPM_RC
TPM2_PolicyAuthorize(
    PolicyAuthorize_In  *in             // IN: input parameter list
    )
{
    SESSION                 *session;
    TPM2B_DIGEST             authHash;
    HASH_STATE               hashState;
    TPMT_TK_VERIFIED         ticket;
    TPM_ALG_ID               hashAlg;
    UINT16                   digestSize;

// Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Extract from the Name of the key, the algorithm used to compute it's Name
    hashAlg = BYTE_ARRAY_TO_UINT16(in->keySign.t.name);

    // 'keySign' parameter needs to use a supported hash algorithm, otherwise
    // can't tell how large the digest should be
    digestSize = CryptGetHashDigestSize(hashAlg);
    if(digestSize == 0)
        return TPM_RC_HASH + RC_PolicyAuthorize_keySign;

    if(digestSize != (in->keySign.t.size - 2))
        return TPM_RC_SIZE + RC_PolicyAuthorize_keySign;

    //If this is a trial policy, skip all validations
    if(session->attributes.isTrialPolicy == CLEAR)
    {
        // Check that "approvedPolicy" matches the current value of the
        // policyDigest in policy session
        if(!Memory2BEqual(&session->u2.policyDigest.b,
                          &in->approvedPolicy.b))
            return TPM_RC_VALUE + RC_PolicyAuthorize_approvedPolicy;

        // Validate ticket TPMT_TK_VERIFIED
        // Compute aHash.  The authorizing object sign a digest
        //  aHash := hash(approvedPolicy || policyRef).
        // Start hash
        authHash.t.size = CryptStartHash(hashAlg, &hashState);

        // add approvedPolicy
        CryptUpdateDigest2B(&hashState, &in->approvedPolicy.b);
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
51
52          // add policyRef
53          CryptUpdateDigest2B(&hashState, &in->policyRef.b);
54
55          // complete hash
56          CryptCompleteHash2B(&hashState, &authHash.b);
57
58          // re-compute TPMT_TK_VERIFIED
59          TicketComputeVerified(in->checkTicket.hierarchy, &authHash,
60                                &in->keySign, &ticket);
61
62          // Compare ticket digest.  If not match, return error
63          if(!Memory2BEqual(&in->checkTicket.digest.b, &ticket.digest.b))
64              return TPM_RC_VALUE+ RC_PolicyAuthorize_checkTicket;
65      }
66
67  // Internal Data Update
68
69      // Set policyDigest to zero digest
70      MemorySet(session->u2.policyDigest.t.buffer, 0,
71                session->u2.policyDigest.t.size);
72
73      // Update policyDigest
74      PolicyContextUpdate(TPM_CC_PolicyAuthorize, &in->keySign, &in->policyRef,
75                          NULL, 0, session);
76
77      return TPM_RC_SUCCESS;
78
79  }
80  #endif // CC_PolicyAuthorize
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.17 TPM2_PolicyAuthValue

24.17.1 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 the command being authorized using 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

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, \text{TPM_CC_PolicyAuthValue}) \quad (36)$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

24.17.2 Command and Response

Table 198 — TPM2_PolicyAuthValue Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyAuthValue
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None

Table 199 — TPM2_PolicyAuthValue Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.17.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyAuthValue_fp.h"
#ifdef TPM_CC_PolicyAuthValue  // Conditional expansion of this file
#include "Policy_spt_fp.h"
TPM_RC
TPM2_PolicyAuthValue(
    PolicyAuthValue_In  *in             // IN: input parameter list
    )
{
    SESSION             *session;
    TPM_CC               commandCode = TPM_CC_PolicyAuthValue;
    HASH_STATE           hashState;

// Internal Data Update

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyAuthValue)
    //   Start hash
    CryptStartHash(session->authHashAlg, &hashState);

    //  add old digest
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

    //  complete the hash and get the results
    CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);

    // update isAuthValueNeeded bit in the session context
    session->attributes.isAuthValueNeeded = SET;
    session->attributes.isPasswordNeeded = CLEAR;

    return TPM_RC_SUCCESS;
}
#endif // CC_PolicyAuthValue
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.18 TPM2_PolicyPassword

24.18.1 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

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, \text{TPM_CC_PolicyAuthValue}) \quad (37)$$

NOTE 2 This is the same extend value as used with TPM2_PolicyAuthValue so that the evaluation can 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.18.2 Command and Response

Table 200 — TPM2_PolicyPassword Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyPassword
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None

Table 201 — TPM2_PolicyPassword Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.18.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyPassword_fp.h"
#ifdef TPM_CC_PolicyPassword  // Conditional expansion of this file
#include "Policy_spt_fp.h"
TPM_RC
TPM2_PolicyPassword(
    PolicyPassword_In   *in             // IN: input parameter list
    )
{
    SESSION             *session;
    TPM_CC               commandCode = TPM_CC_PolicyAuthValue;
    HASH_STATE           hashState;

// Internal Data Update

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyAuthValue)
    //  Start hash
    CryptStartHash(session->authHashAlg, &hashState);

    //  add old digest
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

    //  add commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

    //  complete the digest
    CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);

    //  Update isPasswordNeeded bit
    session->attributes.isPasswordNeeded = SET;
    session->attributes.isAuthValueNeeded = CLEAR;

    return TPM_RC_SUCCESS;
}
#endif // CC_PolicyPassword
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.19 TPM2_PolicyGetDigest

24.19.1 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

24.19.2 Command and Response

Table 202 — TPM2_PolicyGetDigest Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or encrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyGetDigest
TPMI_SH_POLICY	policySession	handle for the policy session Auth Index: None

Table 203 — TPM2_PolicyGetDigest Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	
TPM2B_DIGEST	policyDigest	the current value of the *policySession→policyDigest*

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.19.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyGetDigest_fp.h"
#ifdef TPM_CC_PolicyGetDigest  // Conditional expansion of this file
TPM_RC
TPM2_PolicyGetDigest(
    PolicyGetDigest_In      *in,            // IN: input parameter list
    PolicyGetDigest_Out     *out            // OUT: output parameter list
    )
{
    SESSION     *session;

// Command Output

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    out->policyDigest = session->u2.policyDigest;

    return TPM_RC_SUCCESS;
}
#endif // CC_PolicyGetDigest
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.20 TPM2_PolicyNvWritten

24.20.1 General Description

This command allows a policy to be bound to the TPMA_NV_WRITTEN attributes. This is a deferred assertion. Values are stored in the policy session context and checked when the policy is used for authorization.

If *policySession→checkNVWritten* is CLEAR, it is SET and *policySession→nvWrittenState* is set to *writtenSet*. If *policySession→checkNVWritten* is SET, the TPM will return TPM_RC_VALUE if *policySession→nvWrittenState* and *writtenSet* are not the same.

If the TPM does not return an error, it will update *policySession→policyDigest* by

$$policyDigest_{new} := \mathbf{H}_{policyAlg}(policyDigest_{old} \,||\, \text{TPM_CC_PolicyNvWritten} \,||\, writtenSet) \quad (38)$$

When the policy session is used to authorize a command, the TPM will fail the command if *policySession→checkNVWritten* is SET and *nvIndex→attributes→TPMA_NV_WRITTEN does not match policySession→nvWrittenState.*

NOTE 1 A typical use case is a simple policy for the first write during manufacturing provisioning that would require TPMA_NV_WRITTEN CLEAR and a more complex policy for later use that would require TPMA_NV_WRITTEN SET.

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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.20.2 Command and Response

Table 204 — TPM2_PolicyNvWritten Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	Tag	TPM_ST_SESSIONS if an audit session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_PolicyNVWritten
TPMI_SH_POLICY	policySession	handle for the policy session being extended Auth Index: None
TPMI_YES_NO	writtenSet	YES if NV Index is required to have been written NO if NV Index is required not to have been written

Table 205 — TPM2_PolicyNvWritten Response

Type	Name	Description
TPM_ST	Tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

24.20.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PolicyNvWritten_fp.h"
#ifdef TPM_CC_PolicyNvWritten  // Conditional expansion of this file
```

Make an NV Index policy dependent on the state of the TPMA_NV_WRITTEN attribute of the index.

Table 206 — TPM2_PolicyNvWritten Errors

Error Returns	Meaning
TPM_RC_VALUE	a conflicting request for the attribute has already been processed

```
TPM_RC
TPM2_PolicyNvWritten(
    PolicyNvWritten_In  *in             // IN: input parameter list
    )
{
    SESSION     *session;
    TPM_CC       commandCode = TPM_CC_PolicyNvWritten;
    HASH_STATE   hashState;

// Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // If already set is this a duplicate (the same setting)? If it
    // is a conflicting setting, it is an error
    if(session->attributes.checkNvWritten == SET)
    {
        if((    (session->attributes.nvWrittenState == SET)
            !=  (in->writtenSet == YES)))
            return TPM_RC_VALUE + RC_PolicyNvWritten_writtenSet;
    }

// Internal Data Update

    // Set session attributes so that the NV Index needs to be checked
    session->attributes.checkNvWritten = SET;
    session->attributes.nvWrittenState = (in->writtenSet == YES);

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyNvWritten
    //                        || writtenSet)
    // Start hash
    CryptStartHash(session->authHashAlg, &hashState);

    // add old digest
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

    // add commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

    // add the byte of writtenState
    CryptUpdateDigestInt(&hashState, sizeof(TPMI_YES_NO), &in->writtenSet);

    // complete the digest
    CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);

    return TPM_RC_SUCCESS;
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
53  #endif // CC_PolicyNvWritten
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25 Hierarchy Commands

25.1 TPM2_CreatePrimary

25.1.1 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.

NOTE: Since the sensitive data is not returned, the key cannot be reloaded. It can either be made persistent or it can be recreated.

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*, decrypt, 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 specified in ISO/IEC 11889-1 and implemented in ISO/IEC 11889-4.

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 need to 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*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.1.2 Command and Response

Table 207 — TPM2_CreatePrimary Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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, see ISO/IEC 11889-1, clause 27.3, “Sensitive Values”
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 208 — TPM2_CreatePrimary Response

Type	Name	Description
TPM_ST	tag	see clause 7
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 *creationData* using *nameAlg* of *outPublic*
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

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.1.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "CreatePrimary_fp.h"
#ifdef TPM_CC_CreatePrimary  // Conditional expansion of this file
#include "Object_spt_fp.h"
#include <Platform.h>
```

Table 209 — TPM2_CreatePrimary Errors

Error Returns	Meaning
TPM_RC_ATTRIBUTES	*sensitiveDataOrigin* is CLEAR when 'sensitive. data' is an Empty Buffer, or is SET when 'sensitive. data' is not empty; *fixedTPM*, *fixedParent*, or *encryptedDuplication* attributes are inconsistent between themselves or with those of the parent object; inconsistent *restricted*, *decrypt* and *sign* 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 *decrypt*, *sign*, *restricted* 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;

```
TPM_RC
TPM2_CreatePrimary(
    CreatePrimary_In    *in,            // IN: input parameter list
    CreatePrimary_Out   *out            // OUT: output parameter list
    )
{
// Local variables
    TPM_RC              result = TPM_RC_SUCCESS;
    TPMT_SENSITIVE      sensitive;

// Input Validation
    // The sensitiveDataOrigin attribute must be consistent with the setting of
    // the size of the data object in inSensitive.
    if(  (in->inPublic.t.publicArea.objectAttributes.sensitiveDataOrigin == SET)
       != (in->inSensitive.t.sensitive.data.t.size == 0 ))
        // Mismatch between the object attributes and the parameter.
        return TPM_RC_ATTRIBUTES + RC_CreatePrimary_inSensitive;

    // Check attributes in input public area. TPM_RC_ATTRIBUTES, TPM_RC_KDF,
    // TPM_RC_SCHEME, TPM_RC_SIZE, TPM_RC_SYMMETRIC, or TPM_RC_TYPE error may
    // be returned at this point.
    result = PublicAttributesValidation(FALSE, in->primaryHandle,
                                        &in->inPublic.t.publicArea);
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, RC_CreatePrimary_inPublic);

    // Validate the sensitive area values
    if(  MemoryRemoveTrailingZeros(&in->inSensitive.t.sensitive.userAuth)
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

```
            > CryptGetHashDigestSize(in->inPublic.t.publicArea.nameAlg))
        return TPM_RC_SIZE + RC_CreatePrimary_inSensitive;

// Command output

    // Generate Primary Object
    // The primary key generation process uses the Name of the input public
    // template to compute the key. The keys are generated from the template
    // before anything in the template is allowed to be changed.
    // A TPM_RC_KDF, TPM_RC_SIZE error may be returned at this point
    result = CryptCreateObject(in->primaryHandle, &in->inPublic.t.publicArea,
                               &in->inSensitive.t.sensitive,&sensitive);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Fill in creation data
    FillInCreationData(in->primaryHandle, in->inPublic.t.publicArea.nameAlg,
                       &in->creationPCR, &in->outsideInfo, &out->creationData,
                       &out->creationHash);

    // Copy public area
    out->outPublic = in->inPublic;

    // Fill in private area for output
    ObjectComputeName(&(out->outPublic.t.publicArea), &out->name);

    // Compute creation ticket
    TicketComputeCreation(EntityGetHierarchy(in->primaryHandle), &out->name,
                          &out->creationHash, &out->creationTicket);

    // Create a internal object.  A TPM_RC_OBJECT_MEMORY error may be returned
    //  at this point.
    result = ObjectLoad(in->primaryHandle, &in->inPublic.t.publicArea, &sensitive,
                        &out->name, in->primaryHandle, TRUE, &out->objectHandle);

    return result;
}
#endif // CC_CreatePrimary
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.2 TPM2_HierarchyControl

25.2.1 General Description

This command enables and disables use of a hierarchy and its associated NV storage. The command allows *phEnable*, *phEnableNV*, *shEnable*, and *ehEnable* to be changed when the proper authorization is provided.

This command may be used to CLEAR *phEnable* and *phEnableNV* 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 *phEnable*, *shEnable*, or *ehEnable*, the TPM will disable use of any persistent entity associated with the disabled hierarchy and will flush any transient objects associated with the disabled hierarchy.

When this command is used to CLEAR *shEnable,* the TPM will disable access to any NV index that has TPMA_NV_PLATFORMCREATE CLEAR (indicating that the NV Index was defined using Owner Authorization). As long as *shEnable* is CLEAR, the TPM will return an error in response to any command that attempts to operate upon an NV index that has TPMA_NV_PLATFORMCREATE CLEAR.

When this command is used to CLEAR *phEnableNV*, the TPM will disable access to any NV index that has TPMA_NV_PLATFORMCREATE SET (indicating that the NV Index was defined using Platform Authorization). As long as *phEnableNV* is CLEAR, the TPM will return an error in response to any command that attempts to operate upon an NV index that has TPMA_NV_PLATFORMCREATE SET.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.2.2 Command and Response

Table 210 — TPM2_HierarchyControl Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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_ENABLES	enable	the enable being modified TPM_RH_ENDORSEMENT, TPM_RH_OWNER, TPM_RH_PLATFORM, or TPM_RH_PLATFORM_NV
TPMI_YES_NO	state	YES if the enable should be SET, NO if the enable should be CLEAR

Table 211 — TPM2_HierarchyControl Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.2.3 Detailed Actions

```
1   #include "InternalRoutines.h"
2   #include "HierarchyControl_fp.h"
3   #ifdef TPM_CC_HierarchyControl  // Conditional expansion of this file
```

Table 212 — TPM2_HierarchyControl Errors

Error Returns	Meaning
TPM_RC_AUTH_TYPE	*authHandle* is not applicable to *hierarchy* in its current state

```
4   TPM_RC
5   TPM2_HierarchyControl(
6       HierarchyControl_In     *in             // IN: input parameter list
7       )
8   {
9       TPM_RC      result;
10      BOOL        select = (in->state == YES);
11      BOOL        *selected = NULL;
12
13  // Input Validation
14      switch(in->enable)
15      {
16          // Platform hierarchy has to be disabled by platform auth
17          // If the platform hierarchy has already been disabled, only a reboot
18          // can enable it again
19          case TPM_RH_PLATFORM:
20          case TPM_RH_PLATFORM_NV:
21              if(in->authHandle != TPM_RH_PLATFORM)
22                  return TPM_RC_AUTH_TYPE;
23              break;
24
25          // ShEnable may be disabled if PlatformAuth/PlatformPolicy or
26          // OwnerAuth/OwnerPolicy is provided.  If ShEnable is disabled, then it
27          // may only be enabled if PlatformAuth/PlatformPolicy is provided.
28          case TPM_RH_OWNER:
29              if(   in->authHandle != TPM_RH_PLATFORM
30                 && in->authHandle != TPM_RH_OWNER)
31                  return TPM_RC_AUTH_TYPE;
32              if(   gc.shEnable == FALSE && in->state == YES
33                 && in->authHandle != TPM_RH_PLATFORM)
34                  return TPM_RC_AUTH_TYPE;
35              break;
36
37          // EhEnable may be disabled if either PlatformAuth/PlatformPolicy or
38          // EndosementAuth/EndorsementPolicy is provided.  If EhEnable is disabled,
39          // then it may only be enabled if PlatformAuth/PlatformPolicy is
40          // provided.
41          case TPM_RH_ENDORSEMENT:
42              if(   in->authHandle != TPM_RH_PLATFORM
43                 && in->authHandle != TPM_RH_ENDORSEMENT)
44                  return TPM_RC_AUTH_TYPE;
45              if(   gc.ehEnable == FALSE && in->state == YES
46                 && in->authHandle != TPM_RH_PLATFORM)
47                  return TPM_RC_AUTH_TYPE;
48              break;
49          default:
50              pAssert(FALSE);
51              break;
52      }
53
54  // Internal Data Update
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
55
56      // Enable or disable the selected hierarchy
57      // Note: the authorization processing for this command may keep these
58      // command actions from being executed. For example, if phEnable is
59      // CLEAR, then platformAuth cannot be used for authorization. This
60      // means that would not be possible to use platformAuth to change the
61      // state of phEnable from CLEAR to SET.
62      // If it is decided that platformPolicy can still be used when phEnable
63      // is CLEAR, then this code could SET phEnable when proper platform
64      // policy is provided.
65      switch(in->enable)
66      {
67          case TPM_RH_OWNER:
68              selected = &gc.shEnable;
69              break;
70          case TPM_RH_ENDORSEMENT:
71              selected = &gc.ehEnable;
72              break;
73          case TPM_RH_PLATFORM:
74              selected = &g_phEnable;
75              break;
76          case TPM_RH_PLATFORM_NV:
77              selected = &gc.phEnableNV;
78              break;
79          default:
80              pAssert(FALSE);
81              break;
82      }
83      if(selected != NULL && *selected != select)
84      {
85          // Before changing the internal state, make sure that NV is available.
86          // Only need to update NV if changing the orderly state
87          if(gp.orderlyState != SHUTDOWN_NONE)
88          {
89              // The command needs NV update. Check if NV is available.
90              // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
91              // this point
92              result = NvIsAvailable();
93              if(result != TPM_RC_SUCCESS)
94                  return result;
95          }
96          // state is changing and NV is available so modify
97          *selected = select;
98          // If a hierarchy was just disabled, flush it
99          if(select == CLEAR && in->enable != TPM_RH_PLATFORM_NV)
100         // Flush hierarchy
101             ObjectFlushHierarchy(in->enable);
102
103         // orderly state should be cleared because of the update to state clear data
104         // This gets processed in ExecuteCommand() on the way out.
105         g_clearOrderly = TRUE;
106     }
107     return TPM_RC_SUCCESS;
108 }
109 #endif // CC_HierarchyControl
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.3 TPM2_SetPrimaryPolicy

25.3.1 General Description

This command allows setting of the authorization policy for the lockout (*lockoutPolicy*), 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.3.2 Command and Response

Table 213 — TPM2_SetPrimaryPolicy Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_SetPrimaryPolicy {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_DIGEST	authPolicy	an authorization policy digest; may be the Empty Buffer If *hashAlg* 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 *authPolicy* is an Empty Buffer, then this field shall be TPM_ALG_NULL.

Table 214 — TPM2_SetPrimaryPolicy Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.3.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "SetPrimaryPolicy_fp.h"
#ifdef TPM_CC_SetPrimaryPolicy  // Conditional expansion of this file
```

Table 215 — TPM2_SetPrimaryPolicy Errors

Error Returns	Meaning
TPM_RC_SIZE	size of input *authPolicy* is not consistent with input hash algorithm

```
TPM_RC
TPM2_SetPrimaryPolicy(
    SetPrimaryPolicy_In     *in             // IN: input parameter list
    )
{
    TPM_RC                  result;

// Input Validation

    // Check the authPolicy consistent with hash algorithm. If the policy size is
    // zero, then the algorithm is required to be TPM_ALG_NULL
    if(in->authPolicy.t.size != CryptGetHashDigestSize(in->hashAlg))
        return TPM_RC_SIZE + RC_SetPrimaryPolicy_authPolicy;

    // The command need NV update for OWNER and ENDORSEMENT hierarchy, and
    // might need orderlyState update for PLATFROM hierarchy.
    // Check if NV is available.  A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE
    // error may be returned at this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS)
        return result;

// Internal Data Update

    // Set hierarchy policy
    switch(in->authHandle)
    {
        case TPM_RH_OWNER:
            gp.ownerAlg = in->hashAlg;
            gp.ownerPolicy = in->authPolicy;
            NvWriteReserved(NV_OWNER_ALG, &gp.ownerAlg);
            NvWriteReserved(NV_OWNER_POLICY, &gp.ownerPolicy);
            break;
        case TPM_RH_ENDORSEMENT:
            gp.endorsementAlg = in->hashAlg;
            gp.endorsementPolicy = in->authPolicy;
            NvWriteReserved(NV_ENDORSEMENT_ALG, &gp.endorsementAlg);
            NvWriteReserved(NV_ENDORSEMENT_POLICY, &gp.endorsementPolicy);
            break;
        case TPM_RH_PLATFORM:
            gc.platformAlg = in->hashAlg;
            gc.platformPolicy = in->authPolicy;
            // need to update orderly state
            g_clearOrderly = TRUE;
            break;
        case TPM_RH_LOCKOUT:
            gp.lockoutAlg = in->hashAlg;
            gp.lockoutPolicy = in->authPolicy;
            NvWriteReserved(NV_LOCKOUT_ALG, &gp.lockoutAlg);
            NvWriteReserved(NV_LOCKOUT_POLICY, &gp.lockoutPolicy);
            break;
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
55
56          default:
57              pAssert(FALSE);
58              break;
59      }
60
61      return TPM_RC_SUCCESS;
62  }
63  #endif // CC_SetPrimaryPolicy
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.4 TPM2_ChangePPS

25.4.1 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 Platform Authorization is not changed.

All resident 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 can have configuration information that will be the same after the PPS changes. The Platform can remove the indexes that are no longer needed using TPM2_NV_UndefineSpace().

This command requires Platform Authorization.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.4.2 Command and Response

Table 216 — TPM2_ChangePPS Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 217 — TPM2_ChangePPS Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.4.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "ChangePPS_fp.h"
#ifdef TPM_CC_ChangePPS  // Conditional expansion of this file
TPM_RC
TPM2_ChangePPS(
    ChangePPS_In    *in             // IN: input parameter list
    )
{
    UINT32          i;
    TPM_RC          result;

    // Check if NV is available.  A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE
    // error may be returned at this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

    // Input parameter is not reference in command action
    in = NULL;

// Internal Data Update

    // Reset platform hierarchy seed from RNG
    CryptGenerateRandom(PRIMARY_SEED_SIZE, gp.PPSeed.t.buffer);

    // Create a new phProof value from RNG to prevent the saved platform
    // hierarchy contexts being loaded
    CryptGenerateRandom(PROOF_SIZE, gp.phProof.t.buffer);

    // Set platform authPolicy to null
    gc.platformAlg = TPM_ALG_NULL;
    gc.platformPolicy.t.size = 0;

    // Flush loaded object in platform hierarchy
    ObjectFlushHierarchy(TPM_RH_PLATFORM);

    // Flush platform evict object and index in NV
    NvFlushHierarchy(TPM_RH_PLATFORM);

    // Save hierarchy changes to NV
    NvWriteReserved(NV_PP_SEED, &gp.PPSeed);
    NvWriteReserved(NV_PH_PROOF, &gp.phProof);

    // Re-initialize PCR policies
    for(i = 0; i < NUM_POLICY_PCR_GROUP; i++)
    {
        gp.pcrPolicies.hashAlg[i] = TPM_ALG_NULL;
        gp.pcrPolicies.policy[i].t.size = 0;
    }
    NvWriteReserved(NV_PCR_POLICIES, &gp.pcrPolicies);

    // orderly state should be cleared because of the update to state clear data
    g_clearOrderly = TRUE;

    return TPM_RC_SUCCESS;
}
#endif // CC_ChangePPS
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.5 TPM2_ChangeEPS

25.5.1 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 resident objects (transient or persistent) 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 possible 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 Platform Authorization.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

25.5.2 Command and Response

Table 218 — TPM2_ChangeEPS Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 219 — TPM2_ChangeEPS Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.5.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "ChangeEPS_fp.h"
#ifdef TPM_CC_ChangeEPS  // Conditional expansion of this file
TPM_RC
TPM2_ChangeEPS(
    ChangeEPS_In    *in             // IN: input parameter list
    )
{
    TPM_RC          result;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

    // Input parameter is not reference in command action
    in = NULL;

// Internal Data Update

    // Reset endorsement hierarchy seed from RNG
    CryptGenerateRandom(PRIMARY_SEED_SIZE, gp.EPSeed.t.buffer);

    // Create new ehProof value from RNG
    CryptGenerateRandom(PROOF_SIZE, gp.ehProof.t.buffer);

    // Enable endorsement hierarchy
    gc.ehEnable = TRUE;

    // set authValue buffer to zeros
    MemorySet(gp.endorsementAuth.t.buffer, 0, gp.endorsementAuth.t.size);
    // Set endorsement authValue to null
    gp.endorsementAuth.t.size = 0;

    // Set endorsement authPolicy to null
    gp.endorsementAlg = TPM_ALG_NULL;
    gp.endorsementPolicy.t.size = 0;

    // Flush loaded object in endorsement hierarchy
    ObjectFlushHierarchy(TPM_RH_ENDORSEMENT);

    // Flush evict object of endorsement hierarchy stored in NV
    NvFlushHierarchy(TPM_RH_ENDORSEMENT);

    // Save hierarchy changes to NV
    NvWriteReserved(NV_EP_SEED, &gp.EPSeed);
    NvWriteReserved(NV_EH_PROOF, &gp.ehProof);
    NvWriteReserved(NV_ENDORSEMENT_AUTH, &gp.endorsementAuth);
    NvWriteReserved(NV_ENDORSEMENT_ALG, &gp.endorsementAlg);
    NvWriteReserved(NV_ENDORSEMENT_POLICY, &gp.endorsementPolicy);

    // orderly state should be cleared because of the update to state clear data
    g_clearOrderly = TRUE;

    return TPM_RC_SUCCESS;
}
#endif // CC_ChangeEPS
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.6 TPM2_Clear

25.6.1 General Description

This command removes all TPM context associated with a specific Owner.

The clear operation will:

- flush resident 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 can be set from the RNG or derived from the associated new Primary Seed. If derived from the Primary Seeds, the derivation of *ehProof* needs to use both the SPS and EPS. The computation shall use the SPS as an HMAC key and the derived value can 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, endorsementPolicy,* and *lockoutPolicy* to the Empty Buffer;
- set *Clock* to zero;
- set *resetCount* to zero;
- set *restartCount* to zero; and
- set *Safe* to YES.

This command requires Platform Authorization or Lockout Authorization. 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.6.2 Command and Response

Table 220 — TPM2_Clear Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 221 — TPM2_Clear Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.6.3 Detailed Actions

```
#include "InternalRoutines.h"
#ifdef TPM_CC_Clear  // Conditional expansion of this file
```

Table 222 — TPM2_Clear Errors

Error Returns	Meaning
TPM_RC_DISABLED	Clear command has been disabled

```
TPM_RC
TPM2_Clear(
    Clear_In        *in             // IN: input parameter list
    )
{
    TPM_RC              result;

    // Input parameter is not reference in command action
    in = NULL;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

// Input Validation

    // If Clear command is disabled, return an error
    if(gp.disableClear)
        return TPM_RC_DISABLED;

// Internal Data Update

    // Reset storage hierarchy seed from RNG
    CryptGenerateRandom(PRIMARY_SEED_SIZE, gp.SPSeed.t.buffer);

    // Create new shProof and ehProof value from RNG
    CryptGenerateRandom(PROOF_SIZE, gp.shProof.t.buffer);
    CryptGenerateRandom(PROOF_SIZE, gp.ehProof.t.buffer);

    // Enable storage and endorsement hierarchy
    gc.shEnable = gc.ehEnable = TRUE;

    // set the authValue buffers to zero
    MemorySet(gp.ownerAuth.t.buffer, 0, gp.ownerAuth.t.size);
    MemorySet(gp.endorsementAuth.t.buffer, 0, gp.endorsementAuth.t.size);
    MemorySet(gp.lockoutAuth.t.buffer, 0, gp.lockoutAuth.t.size);
    // Set storage, endorsement and lockout authValue to null
    gp.ownerAuth.t.size = gp.endorsementAuth.t.size = gp.lockoutAuth.t.size = 0;

    // Set storage, endorsement, and lockout authPolicy to null
    gp.ownerAlg = gp.endorsementAlg = gp.lockoutAlg = TPM_ALG_NULL;
    gp.ownerPolicy.t.size = 0;
    gp.endorsementPolicy.t.size = 0;
    gp.lockoutPolicy.t.size = 0;

    // Flush loaded object in storage and endorsement hierarchy
    ObjectFlushHierarchy(TPM_RH_OWNER);
    ObjectFlushHierarchy(TPM_RH_ENDORSEMENT);

```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

```
55          // Flush owner and endorsement object and owner index in NV
56          NvFlushHierarchy(TPM_RH_OWNER);
57          NvFlushHierarchy(TPM_RH_ENDORSEMENT);
58
59          // Save hierarchy changes to NV
60          NvWriteReserved(NV_SP_SEED, &gp.SPSeed);
61          NvWriteReserved(NV_SH_PROOF, &gp.shProof);
62          NvWriteReserved(NV_EH_PROOF, &gp.ehProof);
63          NvWriteReserved(NV_OWNER_AUTH, &gp.ownerAuth);
64          NvWriteReserved(NV_ENDORSEMENT_AUTH, &gp.endorsementAuth);
65          NvWriteReserved(NV_LOCKOUT_AUTH, &gp.lockoutAuth);
66          NvWriteReserved(NV_OWNER_ALG, &gp.ownerAlg);
67          NvWriteReserved(NV_ENDORSEMENT_ALG, &gp.endorsementAlg);
68          NvWriteReserved(NV_LOCKOUT_ALG, &gp.lockoutAlg);
69          NvWriteReserved(NV_OWNER_POLICY, &gp.ownerPolicy);
70          NvWriteReserved(NV_ENDORSEMENT_POLICY, &gp.endorsementPolicy);
71          NvWriteReserved(NV_LOCKOUT_POLICY, &gp.lockoutPolicy);
72
73          // Initialize dictionary attack parameters
74          DAPreInstall_Init();
75
76          // Reset clock
77          go.clock = 0;
78          go.clockSafe = YES;
79          // Update the DRBG state whenever writing orderly state to NV
80          CryptDrbgGetPutState(GET_STATE);
81          NvWriteReserved(NV_ORDERLY_DATA, &go);
82
83          // Reset counters
84          gp.resetCount = gr.restartCount = gr.clearCount = 0;
85          gp.auditCounter = 0;
86          NvWriteReserved(NV_RESET_COUNT, &gp.resetCount);
87          NvWriteReserved(NV_AUDIT_COUNTER, &gp.auditCounter);
88
89          // orderly state should be cleared because of the update to state clear data
90          g_clearOrderly = TRUE;
91
92          return TPM_RC_SUCCESS;
93      }
94      #endif // CC_Clear
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.7 TPM2_ClearControl

25.7.1 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 Platform Authorization to CLEAR the *disableClear* attribute and then execute TPM2_Clear().

Lockout Authorization may be used to SET *disableClear* but not to CLEAR it.

Platform Authorization may be used to SET or CLEAR *disableClear*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.7.2 Command and Response

Table 223 — TPM2_ClearControl Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 *disableOwnerClear* flag is to be SET, NO if the flag is to be CLEAR.

Table 224 — TPM2_ClearControl Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.7.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "ClearControl_fp.h"
#ifdef TPM_CC_ClearControl  // Conditional expansion of this file
```

Table 225 — TPM2_ClearControl Errors

Error Returns	Meaning
TPM_RC_AUTH_FAIL	authorization is not properly given

```
TPM_RC
TPM2_ClearControl(
    ClearControl_In     *in             // IN: input parameter list
    )
{
    TPM_RC      result;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

// Input Validation

    // LockoutAuth may be used to set disableLockoutClear to TRUE but not to FALSE
    if(in->auth == TPM_RH_LOCKOUT && in->disable == NO)
        return TPM_RC_AUTH_FAIL;

// Internal Data Update

    if(in->disable == YES)
        gp.disableClear = TRUE;
    else
        gp.disableClear = FALSE;

    // Record the change to NV
    NvWriteReserved(NV_DISABLE_CLEAR, &gp.disableClear);

    return TPM_RC_SUCCESS;
}
#endif // CC_ClearControl
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.8 TPM2_HierarchyChangeAuth

25.8.1 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 o*wnerAuth* is changed. If *authHandle* is TPM_RH_ENDORSEMENT, then e*ndorsementAuth* 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 27.2, “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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

25.8.2 Command and Response

Table 226 — TPM2_HierarchyChangeAuth Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 value

Table 227 — TPM2_HierarchyChangeAuth Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

25.8.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "HierarchyChangeAuth_fp.h"
#ifdef TPM_CC_HierarchyChangeAuth  // Conditional expansion of this file
#include "Object_spt_fp.h"
```

Table 228 — TPM2_HierarchyChangeAuth Errors

Error Returns	Meaning
TPM_RC_SIZE	*newAuth* size is greater than that of integrity hash digest

```
TPM_RC
TPM2_HierarchyChangeAuth(
    HierarchyChangeAuth_In  *in             // IN: input parameter list
    )
{
    TPM_RC      result;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

    // Make sure the the auth value is a reasonable size (not larger than
    // the size of the digest produced by the integrity hash. The integrity
    // hash is assumed to produce the longest digest of any hash implemented
    // on the TPM.
    if(  MemoryRemoveTrailingZeros(&in->newAuth)
           > CryptGetHashDigestSize(CONTEXT_INTEGRITY_HASH_ALG))
        return TPM_RC_SIZE + RC_HierarchyChangeAuth_newAuth;

    // Set hierarchy authValue
    switch(in->authHandle)
    {
    case TPM_RH_OWNER:
        gp.ownerAuth = in->newAuth;
        NvWriteReserved(NV_OWNER_AUTH, &gp.ownerAuth);
        break;
    case TPM_RH_ENDORSEMENT:
        gp.endorsementAuth = in->newAuth;
        NvWriteReserved(NV_ENDORSEMENT_AUTH, &gp.endorsementAuth);
        break;
    case TPM_RH_PLATFORM:
        gc.platformAuth = in->newAuth;
        // orderly state should be cleared
        g_clearOrderly = TRUE;
        break;
    case TPM_RH_LOCKOUT:
        gp.lockoutAuth = in->newAuth;
        NvWriteReserved(NV_LOCKOUT_AUTH, &gp.lockoutAuth);
        break;
    default:
        pAssert(FALSE);
        break;
    }

    return TPM_RC_SUCCESS;
}
#endif // CC_HierarchyChangeAuth
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

26 Dictionary Attack Functions

26.1 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).

26.2 TPM2_DictionaryAttackLockReset

26.2.1 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 *lockoutAuth* authorization failure is allowed for this command during a *lockoutRecovery* interval (set using TPM2_DictionaryAttackParameters().

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

26.2.2 Command and Response

Table 229 — TPM2_DictionaryAttackLockReset Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_DictionaryAttackLockReset {NV}
TPMI_RH_LOCKOUT	@lockHandle	TPM_RH_LOCKOUT Auth Index: 1 Auth Role: USER

Table 230 — TPM2_DictionaryAttackLockReset Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

26.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "DictionaryAttackLockReset_fp.h"
#ifdef TPM_CC_DictionaryAttackLockReset  // Conditional expansion of this file
TPM_RC
TPM2_DictionaryAttackLockReset(
    DictionaryAttackLockReset_In    *in             // IN: input parameter list
    )
{
    TPM_RC          result;

    // Input parameter is not reference in command action
    in = NULL;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

// Internal Data Update

    // Set failed tries to 0
    gp.failedTries = 0;

    // Record the changes to NV
    NvWriteReserved(NV_FAILED_TRIES, &gp.failedTries);

    return TPM_RC_SUCCESS;
}
#endif // CC_DictionaryAttackLockReset
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

26.3 TPM2_DictionaryAttackParameters

26.3.1 General Description

This command changes the lockout parameters.

The command requires Lockout Authorization.

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 needs to 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 *lockoutAuth* authorization failure is allowed for this command during a *lockoutRecovery* interval.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

26.3.2 Command and Response

Table 231 — TPM2_DictionaryAttackParameters Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 *lockoutAuth* failure before use of *lockoutAuth* is allowed A value of zero indicates that a reboot is required.

Table 232 — TPM2_DictionaryAttackParameters Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

26.3.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "DictionaryAttackParameters_fp.h"
#ifdef TPM_CC_DictionaryAttackParameters  // Conditional expansion of this file
TPM_RC
TPM2_DictionaryAttackParameters(
    DictionaryAttackParameters_In   *in             // IN: input parameter list
    )
{
    TPM_RC          result;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

// Internal Data Update

    // Set dictionary attack parameters
    gp.maxTries = in->newMaxTries;
    gp.recoveryTime = in->newRecoveryTime;
    gp.lockoutRecovery = in->lockoutRecovery;

    // Set failed tries to 0
    gp.failedTries = 0;

    // Record the changes to NV
    NvWriteReserved(NV_FAILED_TRIES, &gp.failedTries);
    NvWriteReserved(NV_MAX_TRIES, &gp.maxTries);
    NvWriteReserved(NV_RECOVERY_TIME, &gp.recoveryTime);
    NvWriteReserved(NV_LOCKOUT_RECOVERY, &gp.lockoutRecovery);

    return TPM_RC_SUCCESS;
}
#endif // CC_DictionaryAttackParameters
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

27 Miscellaneous Management Functions

27.1 Introduction

Clause 27 contains commands that do not logically group with any other commands.

27.2 TPM2_PP_Commands

27.2.1 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.

EXAMPLE *setList* is processed first.

Only commands with handle types of TPMI_RH_PLATFORM, TPMI_RH_PROVISION, TPMI_RH_CLEAR, or TPMI_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 can be made a requirement of any policy.

TPM2_PP_Commands() always requires assertion of Physical Presence.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

27.2.2 Command and Response

Table 233 — TPM2_PP_Commands Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 234 — TPM2_PP_Commands Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

27.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "PP_Commands_fp.h"
#ifdef TPM_CC_PP_Commands  // Conditional expansion of this file
TPM_RC
TPM2_PP_Commands(
    PP_Commands_In  *in             // IN: input parameter list
    )
{
    UINT32          i;

    TPM_RC      result;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

// Internal Data Update

    // Process set list
    for(i = 0; i < in->setList.count; i++)
        // If command is implemented, set it as PP required.  If the input
        // command is not a PP command, it will be ignored at
        // PhysicalPresenceCommandSet().
        if(CommandIsImplemented(in->setList.commandCodes[i]))
            PhysicalPresenceCommandSet(in->setList.commandCodes[i]);

    // Process clear list
    for(i = 0; i < in->clearList.count; i++)
        // If command is implemented, clear it as PP required.  If the input
        // command is not a PP command, it will be ignored at
        // PhysicalPresenceCommandClear().  If the input command is
        // TPM2_PP_Commands, it will be ignored as well
        if(CommandIsImplemented(in->clearList.commandCodes[i]))
            PhysicalPresenceCommandClear(in->clearList.commandCodes[i]);

    // Save the change of PP list
    NvWriteReserved(NV_PP_LIST, &gp.ppList);

    return TPM_RC_SUCCESS;
}
#endif // CC_PP_Commands
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

27.3 TPM2_SetAlgorithmSet

27.3.1 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

27.3.2 Command and Response

Table 235 — TPM2_SetAlgorithmSet Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 236 — TPM2_SetAlgorithmSet Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

27.3.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "SetAlgorithmSet_fp.h"
#ifdef TPM_CC_SetAlgorithmSet  // Conditional expansion of this file
TPM_RC
TPM2_SetAlgorithmSet(
    SetAlgorithmSet_In  *in             // IN: input parameter list
    )
{
    TPM_RC      result;

    // The command needs NV update.  Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

// Internal Data Update
    gp.algorithmSet = in->algorithmSet;

    // Write the algorithm set changes to NV
    NvWriteReserved(NV_ALGORITHM_SET, &gp.algorithmSet);

    return TPM_RC_SUCCESS;
}
#endif // CC_SetAlgorithmSet
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

28 Field Upgrade

28.1 Introduction

Clause 28 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 upgrade process can 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 could 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_FieldUpdateData(). 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

NOTE 2 The TPM needs 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 returned 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 can 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

28.2 TPM2_FieldUpgradeStart

28.2.1 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

28.2.2 Command and Response

Table 237 — TPM2_FieldUpgradeStart Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS
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 *manifestSignature* Auth Index: None
TPM2B_DIGEST	fuDigest	digest of the first block in the field upgrade sequence
TPMT_SIGNATURE	manifestSignature	signature over *fuDigest* using the key associated with *keyHandle* (not optional)

Table 238 — TPM2_FieldUpgradeStart Response

Type	Name	Description
TPM_ST	tag	see clause 7
UINT32	responseSize	
TPM_RC	responseCode	

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

28.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "FieldUpgradeStart_fp.h"
#ifdef TPM_CC_FieldUpgradeStart  // Conditional expansion of this file
TPM_RC
TPM2_FieldUpgradeStart(
    FieldUpgradeStart_In    *in             // IN: input parameter list
    )
{
    // Not implemented
    UNUSED_PARAMETER(in);
    return TPM_RC_SUCCESS;
}
#endif
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

28.3 TPM2_FieldUpgradeData

28.3.1 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

 © ISO/IEC 2015 – All rights reserved

28.3.2 Command and Response

Table 239 — TPM2_FieldUpgradeData Command

Type	Name	Description
TPMI_ST_COMMAND_TAG	tag	TPM_ST_SESSIONS if an audit or decrypt session is present; otherwise, TPM_ST_NO_SESSIONS
UINT32	commandSize	
TPM_CC	commandCode	TPM_CC_FieldUpgradeData {NV}
TPM2B_MAX_BUFFER	fuData	field upgrade image data

Table 240 — TPM2_FieldUpgradeData Response

Type	Name	Description
TPM_ST	tag	see clause 7
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

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

28.3.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "FieldUpgradeData_fp.h"
#ifdef TPM_CC_FieldUpgradeData  // Conditional expansion of this file
TPM_RC
TPM2_FieldUpgradeData(
    FieldUpgradeData_In     *in,            // IN: input parameter list
    FieldUpgradeData_Out    *out            // OUT: output parameter list
    )
{
    // Not implemented
    UNUSED_PARAMETER(in);
    UNUSED_PARAMETER(out);
    return TPM_RC_SUCCESS;
}
#endif
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved

28.4 TPM2_FirmwareRead

28.4.1 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 ought to 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().

IECNORM.COM : Click to view the full PDF of ISO/IEC 11889-3:2015

© ISO/IEC 2015 – All rights reserved