Trusted Platform Module Library



Part 2: Structures

Family “2.0”

Level 00 Revision 01.59

November 8, 2019

Committee Draft

Contact: admin@trustedcomputinggroup.org

TCG Confidential

Copyright © TCG 2006-2019

**Licenses and Notices**

**Copyright Licenses**:

Trusted Computing Group (TCG) grants to the user of the source code in this specification (the “Source Code”) a worldwide, irrevocable, nonexclusive, royalty free, copyright license to reproduce, create derivative works, distribute, display and perform the Source Code and derivative works thereof, and to grant others the rights granted herein.

The TCG grants to the user of the other parts of the specification (other than the Source Code) the rights to reproduce, distribute, display, and perform the specification solely for the purpose of developing products based on such documents.

**Source Code Distribution Conditions**:

Redistributions of Source Code must retain the above copyright licenses, this list of conditions and the following disclaimers.

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

**Disclaimers**:

THE COPYRIGHT LICENSES SET FORTH ABOVE DO NOT REPRESENT ANY FORM OF LICENSE OR WAIVER, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, WITH RESPECT TO PATENT RIGHTS HELD BY TCG MEMBERS (OR OTHER THIRD PARTIES) THAT MAY BE NECESSARY TO IMPLEMENT THIS SPECIFICATION OR OTHERWISE. Contact TCG Administration ([admin@trustedcomputinggroup.org](mailto:admin@trustedcomputinggroup.org)) for information on specification licensing rights available through TCG membership agreements.

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, ACCURACY, COMPLETENESS, OR NONINFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

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

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

CONTENTS

[1 Scope 1](#_Toc20317506)

[2 Terms and definitions 1](#_Toc20317507)

[3 Symbols and abbreviated terms 1](#_Toc20317508)

[4 Notation 1](#_Toc20317509)

[4.1 Introduction 1](#_Toc20317510)

[4.2 Named Constants 2](#_Toc20317511)

[4.3 Data Type Aliases (typedefs) 3](#_Toc20317512)

[4.4 Enumerations 3](#_Toc20317513)

[4.5 Interface Type 4](#_Toc20317514)

[4.6 Arrays 5](#_Toc20317515)

[4.7 Structure Definitions 6](#_Toc20317516)

[4.8 Conditional Types 7](#_Toc20317517)

[4.9 Unions 9](#_Toc20317518)

[4.9.1 Introduction 9](#_Toc20317519)

[4.9.2 Union Definition 9](#_Toc20317520)

[4.9.3 Union Instance 10](#_Toc20317521)

[4.9.4 Union Selector Definition 11](#_Toc20317522)

[4.10 Bit Field Definitions 12](#_Toc20317523)

[4.11 Parameter Limits 13](#_Toc20317524)

[4.12 Algorithm Macros 14](#_Toc20317525)

[4.12.1 Introduction 14](#_Toc20317526)

[4.12.2 Algorithm Token Semantics 15](#_Toc20317527)

[4.12.3 Algorithm Tokens in Unions 15](#_Toc20317528)

[4.12.4 Algorithm Tokens in Interface Types 16](#_Toc20317529)

[4.12.5 Algorithm Tokens for Table Replication 16](#_Toc20317530)

[4.13 Size Checking 18](#_Toc20317531)

[4.14 Data Direction 18](#_Toc20317532)

[4.15 Structure Validations 20](#_Toc20317533)

[4.16 Name Prefix Convention 20](#_Toc20317534)

[4.17 Data Alignment 21](#_Toc20317535)

[4.18 Parameter Unmarshaling Errors 21](#_Toc20317536)

[5 Base Types 23](#_Toc20317537)

[5.1 Primitive Types 23](#_Toc20317538)

[5.2 Specification Logic Value Constants 23](#_Toc20317539)

[5.3 Miscellaneous Types 24](#_Toc20317540)

[6 Constants 25](#_Toc20317541)

[6.1 TPM\_SPEC (Specification Version Values) 25](#_Toc20317542)

[6.2 TPM\_GENERATED 25](#_Toc20317543)

[6.3 TPM\_ALG\_ID 26](#_Toc20317544)

[6.4 TPM\_ECC\_CURVE 30](#_Toc20317545)

[6.5 TPM\_CC (Command Codes) 30](#_Toc20317546)

[6.5.1 Format 30](#_Toc20317547)

[6.5.2 TPM\_CC Listing 31](#_Toc20317548)

[6.6 TPM\_RC (Response Codes) 35](#_Toc20317549)

[6.6.1 Description 35](#_Toc20317550)

[6.6.2 Response Code Formats 35](#_Toc20317551)

[6.6.3 TPM\_RC Values 38](#_Toc20317552)

[6.7 TPM\_CLOCK\_ADJUST 43](#_Toc20317553)

[6.8 TPM\_EO (EA Arithmetic Operands) 43](#_Toc20317554)

[6.9 TPM\_ST (Structure Tags) 44](#_Toc20317555)

[6.10 TPM\_SU (Startup Type) 46](#_Toc20317556)

[6.11 TPM\_SE (Session Type) 47](#_Toc20317557)

[6.12 TPM\_CAP (Capabilities) 48](#_Toc20317558)

[6.13 TPM\_PT (Property Tag) 49](#_Toc20317559)

[6.14 TPM\_PT\_PCR (PCR Property Tag) 54](#_Toc20317560)

[6.15 TPM\_PS (Platform Specific) 56](#_Toc20317561)

[7 Handles 57](#_Toc20317562)

[7.1 Introduction 57](#_Toc20317563)

[7.2 TPM\_HT (Handle Types) 57](#_Toc20317564)

[7.3 Persistent Handle Sub-ranges 58](#_Toc20317565)

[7.4 TPM\_RH (Permanent Handles) 59](#_Toc20317566)

[7.5 TPM\_HC (Handle Value Constants) 60](#_Toc20317567)

[8 Attribute Structures 63](#_Toc20317568)

[8.1 Description 63](#_Toc20317569)

[8.2 TPMA\_ALGORITHM 63](#_Toc20317570)

[8.3 TPMA\_OBJECT (Object Attributes) 64](#_Toc20317571)

[8.3.1 Introduction 64](#_Toc20317572)

[8.3.2 Structure Definition 64](#_Toc20317573)

[8.3.3 Attribute Descriptions 65](#_Toc20317574)

[8.3.3.1 Introduction 65](#_Toc20317575)

[8.3.3.2 Bit[1] – *fixedTPM* 66](#_Toc20317576)

[8.3.3.3 Bit[2] – *stClear* 66](#_Toc20317577)

[8.3.3.4 Bit[4] – *fixedParent* 66](#_Toc20317578)

[8.3.3.5 Bit[5] – *sensitiveDataOrigin* 66](#_Toc20317579)

[8.3.3.6 Bit[6] – *userWithAuth* 67](#_Toc20317580)

[8.3.3.7 Bit[7] – *adminWithPolicy* 67](#_Toc20317581)

[8.3.3.8 Bit[10] – *noDA* 68](#_Toc20317582)

[8.3.3.9 Bit[11] – *encryptedDuplication* 68](#_Toc20317583)

[8.3.3.10 Bit[16] – *restricted* 69](#_Toc20317584)

[8.3.3.11 Bit[17] – *decrypt* 69](#_Toc20317585)

[8.3.3.12 Bit[18] – *sign* / *encrypt* 70](#_Toc20317586)

[8.3.3.13 Bit[19] – *x509sign* 70](#_Toc20317587)

[8.4 TPMA\_SESSION (Session Attributes) 71](#_Toc20317588)

[8.5 TPMA\_LOCALITY (Locality Attribute) 73](#_Toc20317589)

[8.6 TPMA\_PERMANENT 74](#_Toc20317590)

[8.7 TPMA\_STARTUP\_CLEAR 75](#_Toc20317591)

[8.8 TPMA\_MEMORY 76](#_Toc20317592)

[8.9 TPMA\_CC (Command Code Attributes) 77](#_Toc20317593)

[8.9.1 Introduction 77](#_Toc20317594)

[8.9.2 Structure Definition 77](#_Toc20317595)

[8.9.3 Field Descriptions 77](#_Toc20317596)

[8.9.3.1 Bits[15:0] – *commandIndex* 77](#_Toc20317597)

[8.9.3.2 Bit[22] – *nv* 77](#_Toc20317598)

[8.9.3.3 Bit[23] – *extensive* 78](#_Toc20317599)

[8.9.3.4 Bit[24] – *flushed* 78](#_Toc20317600)

[8.9.3.5 Bits[27:25] – *cHandles* 78](#_Toc20317601)

[8.9.3.6 Bit[28] – *rHandle* 78](#_Toc20317602)

[8.9.3.7 Bit[29] – V 78](#_Toc20317603)

[8.9.3.8 Bits[31:30] – Res 79](#_Toc20317604)

[8.10 TPMA\_MODES 79](#_Toc20317605)

[8.11 TPMA\_X509\_KEY\_USAGE 80](#_Toc20317606)

[8.12 TPMA\_ACT 81](#_Toc20317607)

[9 Interface Types 82](#_Toc20317608)

[9.1 Introduction 82](#_Toc20317609)

[9.2 TPMI\_YES\_NO 82](#_Toc20317610)

[9.3 TPMI\_DH\_OBJECT 82](#_Toc20317611)

[9.4 TPMI\_DH\_PARENT 83](#_Toc20317612)

[9.5 TPMI\_DH\_PERSISTENT 83](#_Toc20317613)

[9.6 TPMI\_DH\_ENTITY 84](#_Toc20317614)

[9.7 TPMI\_DH\_PCR 84](#_Toc20317615)

[9.8 TPMI\_SH\_AUTH\_SESSION 85](#_Toc20317616)

[9.9 TPMI\_SH\_HMAC 85](#_Toc20317617)

[9.10 TPMI\_SH\_POLICY 85](#_Toc20317618)

[9.11 TPMI\_DH\_CONTEXT 85](#_Toc20317619)

[9.12 TPMI\_DH\_SAVED 86](#_Toc20317620)

[9.13 TPMI\_RH\_HIERARCHY 86](#_Toc20317621)

[9.14 TPMI\_RH\_ENABLES 86](#_Toc20317622)

[9.15 TPMI\_RH\_HIERARCHY\_AUTH 87](#_Toc20317623)

[9.16 TPMI\_RH\_HIERARCHY\_POLICY 87](#_Toc20317624)

[9.17 TPMI\_RH\_PLATFORM 87](#_Toc20317625)

[9.18 TPMI\_RH\_OWNER 88](#_Toc20317626)

[9.19 TPMI\_RH\_ENDORSEMENT 88](#_Toc20317627)

[9.20 TPMI\_RH\_PROVISION 88](#_Toc20317628)

[9.21 TPMI\_RH\_CLEAR 89](#_Toc20317629)

[9.22 TPMI\_RH\_NV\_AUTH 89](#_Toc20317630)

[9.23 TPMI\_RH\_LOCKOUT 89](#_Toc20317631)

[9.24 TPMI\_RH\_NV\_INDEX 90](#_Toc20317632)

[9.25 TPMI\_RH\_AC 90](#_Toc20317633)

[9.26 TPMI\_RH\_ACT 91](#_Toc20317634)

[9.27 TPMI\_ALG\_HASH 91](#_Toc20317635)

[9.28 TPMI\_ALG\_ASYM (Asymmetric Algorithms) 91](#_Toc20317636)

[9.29 TPMI\_ALG\_SYM (Symmetric Algorithms) 92](#_Toc20317637)

[9.30 TPMI\_ALG\_SYM\_OBJECT 92](#_Toc20317638)

[9.31 TPMI\_ALG\_SYM\_MODE 92](#_Toc20317639)

[9.32 TPMI\_ALG\_KDF (Key and Mask Generation Functions) 93](#_Toc20317640)

[9.33 TPMI\_ALG\_SIG\_SCHEME 93](#_Toc20317641)

[9.34 TPMI\_ECC\_KEY\_EXCHANGE 93](#_Toc20317642)

[9.35 TPMI\_ST\_COMMAND\_TAG 94](#_Toc20317643)

[9.36 TPMI\_ALG\_MAC\_SCHEME 94](#_Toc20317644)

[9.37 TPMI\_ALG\_CIPHER\_MODE 94](#_Toc20317645)

[10 Structure Definitions 95](#_Toc20317646)

[10.1 TPMS\_EMPTY 95](#_Toc20317647)

[10.2 TPMS\_ALGORITHM\_DESCRIPTION 95](#_Toc20317648)

[10.3 Hash/Digest Structures 95](#_Toc20317649)

[10.3.1 TPMU\_HA (Hash) 95](#_Toc20317650)

[10.3.2 TPMT\_HA 96](#_Toc20317651)

[10.4 Sized Buffers 96](#_Toc20317652)

[10.4.1 Introduction 96](#_Toc20317653)

[10.4.2 TPM2B\_DIGEST 97](#_Toc20317654)

[10.4.3 TPM2B\_DATA 97](#_Toc20317655)

[10.4.4 TPM2B\_NONCE 97](#_Toc20317656)

[10.4.5 TPM2B\_AUTH 97](#_Toc20317657)

[10.4.6 TPM2B\_OPERAND 98](#_Toc20317658)

[10.4.7 TPM2B\_EVENT 98](#_Toc20317659)

[10.4.8 TPM2B\_MAX\_BUFFER 98](#_Toc20317660)

[10.4.9 TPM2B\_MAX\_NV\_BUFFER 98](#_Toc20317661)

[10.4.10 TPM2B\_TIMEOUT 99](#_Toc20317662)

[10.4.11 TPM2B\_IV 99](#_Toc20317663)

[10.5 Names 99](#_Toc20317664)

[10.5.1 Introduction 99](#_Toc20317665)

[10.5.2 TPMU\_NAME 99](#_Toc20317666)

[10.5.3 TPM2B\_NAME 100](#_Toc20317667)

[10.6 PCR Structures 100](#_Toc20317668)

[10.6.1 TPMS\_PCR\_SELECT 100](#_Toc20317669)

[10.6.2 TPMS\_PCR\_SELECTION 101](#_Toc20317670)

[10.7 Tickets 101](#_Toc20317671)

[10.7.1 Introduction 101](#_Toc20317672)

[10.7.2 A NULL Ticket 102](#_Toc20317673)

[10.7.3 TPMT\_TK\_CREATION 102](#_Toc20317674)

[10.7.4 TPMT\_TK\_VERIFIED 103](#_Toc20317675)

[10.7.5 TPMT\_TK\_AUTH 104](#_Toc20317676)

[10.7.6 TPMT\_TK\_HASHCHECK 105](#_Toc20317677)

[10.8 Property Structures 105](#_Toc20317678)

[10.8.1 TPMS\_ALG\_PROPERTY 105](#_Toc20317679)

[10.8.2 TPMS\_TAGGED\_PROPERTY 105](#_Toc20317680)

[10.8.3 TPMS\_TAGGED\_PCR\_SELECT 106](#_Toc20317681)

[10.8.4 TPMS\_TAGGED\_POLICY 106](#_Toc20317682)

[10.8.5 TPMS\_ACT\_DATA 106](#_Toc20317683)

[10.9 Lists 107](#_Toc20317684)

[10.9.1 TPML\_CC 107](#_Toc20317685)

[10.9.2 TPML\_CCA 107](#_Toc20317686)

[10.9.3 TPML\_ALG 107](#_Toc20317687)

[10.9.4 TPML\_HANDLE 108](#_Toc20317688)

[10.9.5 TPML\_DIGEST 108](#_Toc20317689)

[10.9.6 TPML\_DIGEST\_VALUES 109](#_Toc20317690)

[10.9.7 TPML\_PCR\_SELECTION 109](#_Toc20317691)

[10.9.8 TPML\_ALG\_PROPERTY 109](#_Toc20317692)

[10.9.9 TPML\_TAGGED\_TPM\_PROPERTY 110](#_Toc20317693)

[10.9.10 TPML\_TAGGED\_PCR\_PROPERTY 110](#_Toc20317694)

[10.9.11 TPML\_ECC\_CURVE 110](#_Toc20317695)

[10.9.12 TPML\_TAGGED\_POLICY 111](#_Toc20317696)

[10.9.13 TPML\_ACT\_DATA 111](#_Toc20317697)

[10.10 Capabilities Structures 111](#_Toc20317698)

[10.10.1 TPMU\_CAPABILITIES 112](#_Toc20317699)

[10.10.2 TPMS\_CAPABILITY\_DATA 112](#_Toc20317700)

[10.11 Clock/Counter Structures 112](#_Toc20317701)

[10.11.1 TPMS\_CLOCK\_INFO 112](#_Toc20317702)

[10.11.2 *Clock* 113](#_Toc20317703)

[10.11.3 *ResetCount* 113](#_Toc20317704)

[10.11.4 *RestartCount* 113](#_Toc20317705)

[10.11.5 *Safe* 113](#_Toc20317706)

[10.11.6 TPMS\_TIME\_INFO 114](#_Toc20317707)

[10.12 TPM Attestation Structures 114](#_Toc20317708)

[10.12.1 Introduction 114](#_Toc20317709)

[10.12.2 TPMS\_TIME\_ATTEST\_INFO 114](#_Toc20317710)

[10.12.3 TPMS\_CERTIFY\_INFO 114](#_Toc20317711)

[10.12.4 TPMS\_QUOTE\_INFO 115](#_Toc20317712)

[10.12.5 TPMS\_COMMAND\_AUDIT\_INFO 115](#_Toc20317713)

[10.12.6 TPMS\_SESSION\_AUDIT\_INFO 115](#_Toc20317714)

[10.12.7 TPMS\_CREATION\_INFO 115](#_Toc20317715)

[10.12.8 TPMS\_NV\_CERTIFY\_INFO 116](#_Toc20317716)

[10.12.9 TPMS\_NV\_DIGEST\_CERTIFY\_INFO 116](#_Toc20317717)

[10.12.10 TPMI\_ST\_ATTEST 116](#_Toc20317718)

[10.12.11 TPMU\_ATTEST 117](#_Toc20317719)

[10.12.12 TPMS\_ATTEST 117](#_Toc20317720)

[10.12.13 TPM2B\_ATTEST 118](#_Toc20317721)

[10.13 Authorization Structures 118](#_Toc20317722)

[10.13.1 Introduction 118](#_Toc20317723)

[10.13.2 TPMS\_AUTH\_COMMAND 118](#_Toc20317724)

[10.13.3 TPMS\_AUTH\_RESPONSE 118](#_Toc20317725)

[11 Algorithm Parameters and Structures 119](#_Toc20317726)

[11.1 Symmetric 119](#_Toc20317727)

[11.1.1 Introduction 119](#_Toc20317728)

[11.1.2 TPMI\_!ALG.S\_KEY\_BITS 119](#_Toc20317729)

[11.1.3 TPMU\_SYM\_KEY\_BITS 119](#_Toc20317730)

[11.1.4 TPMU\_SYM\_MODE 120](#_Toc20317731)

[11.1.5 TPMU\_SYM\_DETAILS 120](#_Toc20317732)

[11.1.6 TPMT\_SYM\_DEF 121](#_Toc20317733)

[11.1.7 TPMT\_SYM\_DEF\_OBJECT 121](#_Toc20317734)

[11.1.8 TPM2B\_SYM\_KEY 121](#_Toc20317735)

[11.1.9 TPMS\_SYMCIPHER\_PARMS 122](#_Toc20317736)

[11.1.10 TPM2B\_LABEL 122](#_Toc20317737)

[11.1.11 TPMS\_DERIVE 122](#_Toc20317738)

[11.1.12 TPM2B\_DERIVE 123](#_Toc20317739)

[11.1.13 TPMU\_SENSITIVE\_CREATE 123](#_Toc20317740)

[11.1.14 TPM2B\_SENSITIVE\_DATA 123](#_Toc20317741)

[11.1.15 TPMS\_SENSITIVE\_CREATE 124](#_Toc20317742)

[11.1.16 TPM2B\_SENSITIVE\_CREATE 124](#_Toc20317743)

[11.1.17 TPMS\_SCHEME\_HASH 124](#_Toc20317744)

[11.1.18 TPMS\_SCHEME\_ECDAA 125](#_Toc20317745)

[11.1.19 TPMI\_ALG\_KEYEDHASH\_SCHEME 125](#_Toc20317746)

[11.1.20 HMAC\_SIG\_SCHEME 125](#_Toc20317747)

[11.1.21 TPMS\_SCHEME\_XOR 125](#_Toc20317748)

[11.1.22 TPMU\_SCHEME\_KEYEDHASH 126](#_Toc20317749)

[11.1.23 TPMT\_KEYEDHASH\_SCHEME 126](#_Toc20317750)

[11.2 Asymmetric 127](#_Toc20317751)

[11.2.1 Signing Schemes 127](#_Toc20317752)

[11.2.1.1 Introduction 127](#_Toc20317753)

[11.2.1.2 RSA Signature Schemes 127](#_Toc20317754)

[11.2.1.3 ECC Signature Schemes 127](#_Toc20317755)

[11.2.1.4 TPMU\_SIG\_SCHEME 128](#_Toc20317756)

[11.2.1.5 TPMT\_SIG\_SCHEME 128](#_Toc20317757)

[11.2.2 Encryption Schemes 128](#_Toc20317758)

[11.2.2.1 Introduction 128](#_Toc20317759)

[11.2.2.2 RSA Encryption Schemes 128](#_Toc20317760)

[11.2.2.3 ECC Key Exchange Schemes 129](#_Toc20317761)

[11.2.3 Key Derivation Schemes 129](#_Toc20317762)

[11.2.3.1 Introduction 129](#_Toc20317763)

[11.2.3.2 TPMU\_KDF\_SCHEME 129](#_Toc20317764)

[11.2.3.3 TPMT\_KDF\_SCHEME 129](#_Toc20317765)

[11.2.3.4 TPMI\_ALG\_ASYM\_SCHEME 130](#_Toc20317766)

[11.2.3.5 TPMU\_ASYM\_SCHEME 130](#_Toc20317767)

[11.2.3.6 TPMT\_ASYM\_SCHEME 130](#_Toc20317768)

[11.2.4 RSA 131](#_Toc20317769)

[11.2.4.1 TPMI\_ALG\_RSA\_SCHEME 131](#_Toc20317770)

[11.2.4.2 TPMT\_RSA\_SCHEME 131](#_Toc20317771)

[11.2.4.3 TPMI\_ALG\_RSA\_DECRYPT 131](#_Toc20317772)

[11.2.4.4 TPMT\_RSA\_DECRYPT 131](#_Toc20317773)

[11.2.4.5 TPM2B\_PUBLIC\_KEY\_RSA 132](#_Toc20317774)

[11.2.4.6 TPMI\_RSA\_KEY\_BITS 132](#_Toc20317775)

[11.2.4.7 TPM2B\_PRIVATE\_KEY\_RSA 132](#_Toc20317776)

[11.2.5 ECC 133](#_Toc20317777)

[11.2.5.1 TPM2B\_ECC\_PARAMETER 133](#_Toc20317778)

[11.2.5.2 TPMS\_ECC\_POINT 133](#_Toc20317779)

[11.2.5.3 TPM2B\_ECC\_POINT 133](#_Toc20317780)

[11.2.5.4 TPMI\_ALG\_ECC\_SCHEME 134](#_Toc20317781)

[11.2.5.5 TPMI\_ECC\_CURVE 134](#_Toc20317782)

[11.2.5.6 TPMT\_ECC\_SCHEME 134](#_Toc20317783)

[11.2.5.7 TPMS\_ALGORITHM\_DETAIL\_ECC 135](#_Toc20317784)

[11.3 Signatures 135](#_Toc20317785)

[11.3.1 TPMS\_SIGNATURE\_RSA 135](#_Toc20317786)

[11.3.2 TPMS\_SIGNATURE\_ECC 136](#_Toc20317787)

[11.3.3 TPMU\_SIGNATURE 136](#_Toc20317788)

[11.3.4 TPMT\_SIGNATURE 136](#_Toc20317789)

[11.4 Key/Secret Exchange 137](#_Toc20317790)

[11.4.1 Introduction 137](#_Toc20317791)

[11.4.2 TPMU\_ENCRYPTED\_SECRET 137](#_Toc20317792)

[11.4.3 TPM2B\_ENCRYPTED\_SECRET 137](#_Toc20317793)

[12 Key/Object Complex 138](#_Toc20317794)

[12.1 Introduction 138](#_Toc20317795)

[12.2 Public Area Structures 138](#_Toc20317796)

[12.2.1 Description 138](#_Toc20317797)

[12.2.2 TPMI\_ALG\_PUBLIC 138](#_Toc20317798)

[12.2.3 Type-Specific Parameters 139](#_Toc20317799)

[12.2.3.1 Description 139](#_Toc20317800)

[12.2.3.2 TPMU\_PUBLIC\_ID 139](#_Toc20317801)

[12.2.3.3 TPMS\_KEYEDHASH\_PARMS 140](#_Toc20317802)

[12.2.3.4 TPMS\_ASYM\_PARMS 140](#_Toc20317803)

[12.2.3.5 TPMS\_RSA\_PARMS 141](#_Toc20317804)

[12.2.3.6 TPMS\_ECC\_PARMS 142](#_Toc20317805)

[12.2.3.7 TPMU\_PUBLIC\_PARMS 142](#_Toc20317806)

[12.2.3.8 TPMT\_PUBLIC\_PARMS 143](#_Toc20317807)

[12.2.4 TPMT\_PUBLIC 143](#_Toc20317808)

[12.2.5 TPM2B\_PUBLIC 143](#_Toc20317809)

[12.2.6 TPM2B\_TEMPLATE 144](#_Toc20317810)

[12.3 Private Area Structures 144](#_Toc20317811)

[12.3.1 Introduction 144](#_Toc20317812)

[12.3.2 Sensitive Data Structures 144](#_Toc20317813)

[12.3.2.1 Introduction 144](#_Toc20317814)

[12.3.2.2 TPM2B\_PRIVATE\_VENDOR\_SPECIFIC 144](#_Toc20317815)

[12.3.2.3 TPMU\_SENSITIVE\_COMPOSITE 145](#_Toc20317816)

[12.3.2.4 TPMT\_SENSITIVE 145](#_Toc20317817)

[12.3.3 TPM2B\_SENSITIVE 145](#_Toc20317818)

[12.3.4 Encryption 146](#_Toc20317819)

[12.3.5 Integrity 146](#_Toc20317820)

[12.3.6 \_PRIVATE 146](#_Toc20317821)

[12.3.7 TPM2B\_PRIVATE 146](#_Toc20317822)

[12.4 Identity Object 147](#_Toc20317823)

[12.4.1 Description 147](#_Toc20317824)

[12.4.2 TPMS\_ID\_OBJECT 147](#_Toc20317825)

[12.4.3 TPM2B\_ID\_OBJECT 147](#_Toc20317826)

[13 NV Storage Structures 148](#_Toc20317827)

[13.1 TPM\_NV\_INDEX 148](#_Toc20317828)

[13.2 TPM\_NT 149](#_Toc20317829)

[13.3 TPMS\_NV\_PIN\_COUNTER\_PARAMETERS 149](#_Toc20317830)

[13.4 TPMA\_NV (NV Index Attributes) 149](#_Toc20317831)

[13.5 TPMS\_NV\_PUBLIC 153](#_Toc20317832)

[13.6 TPM2B\_NV\_PUBLIC 153](#_Toc20317833)

[14 Context Data 154](#_Toc20317834)

[14.1 Introduction 154](#_Toc20317835)

[14.2 TPM2B\_CONTEXT\_SENSITIVE 154](#_Toc20317836)

[14.3 TPMS\_CONTEXT\_DATA 154](#_Toc20317837)

[14.4 TPM2B\_CONTEXT\_DATA 154](#_Toc20317838)

[14.5 TPMS\_CONTEXT 155](#_Toc20317839)

[14.6 Parameters of TPMS\_CONTEXT 156](#_Toc20317840)

[14.6.1 *sequence* 156](#_Toc20317841)

[14.6.2 *savedHandle* 156](#_Toc20317842)

[14.6.3 *hierarchy* 157](#_Toc20317843)

[14.7 Context Protection 157](#_Toc20317844)

[14.7.1 Context Integrity 157](#_Toc20317845)

[14.7.2 Context Confidentiality 157](#_Toc20317846)

[15 Creation Data 158](#_Toc20317847)

[15.1 TPMS\_CREATION\_DATA 158](#_Toc20317848)

[15.2 TPM2B\_CREATION\_DATA 158](#_Toc20317849)

[16 Attached Component Structures 159](#_Toc20317850)

[16.1 TPM\_AT 159](#_Toc20317851)

[16.2 TPM\_AE 159](#_Toc20317852)

[16.3 TPMS\_AC\_OUTPUT 159](#_Toc20317853)

[16.4 TPML\_AC\_CAPABILITIES 160](#_Toc20317854)

Tables

[Table 1 — Name Prefix Convention 20](#_Toc21376048)

[Table 2 — Unmarshaling Errors 22](#_Toc21376049)

[Table 3 — Definition of Base Types 23](#_Toc21376050)

[Table 4 — Defines for Logic Values 23](#_Toc21376051)

[Table 5 — Definition of Types for Documentation Clarity 24](#_Toc21376052)

[Table 6 — Definition of (UINT32) TPM\_SPEC Constants <> 25](#_Toc21376053)

[Table 7 — Definition of (UINT32) TPM\_GENERATED Constants <O> 25](#_Toc21376054)

[Table 8 — Legend for TPM\_ALG\_ID Table 26](#_Toc21376055)

[Table 9 — Definition of (UINT16) TPM\_ALG\_ID Constants <IN/OUT, S> 27](#_Toc21376056)

[Table 10 — Definition of (UINT16) {ECC} TPM\_ECC\_CURVE Constants <IN/OUT> 30](#_Toc21376057)

[Table 11 — TPM Command Format Fields Description 30](#_Toc21376058)

[Table 12 — Definition of (UINT32) TPM\_CC Constants (Numeric Order) <IN/OUT, S> 31](#_Toc21376059)

[Table 13 — Format-Zero Response Codes 36](#_Toc21376060)

[Table 14 — Format-One Response Codes 37](#_Toc21376061)

[Table 15 — Response Code Groupings 37](#_Toc21376062)

[Table 16 — Definition of (UINT32) TPM\_RC Constants (Actions) <OUT> 38](#_Toc21376063)

[Table 17 — Definition of (INT8) TPM\_CLOCK\_ADJUST Constants <IN> 43](#_Toc21376064)

[Table 18 — Definition of (UINT16) TPM\_EO Constants <IN/OUT> 43](#_Toc21376065)

[Table 19 — Definition of (UINT16) TPM\_ST Constants <IN/OUT, S> 45](#_Toc21376066)

[Table 20 — Definition of (UINT16) TPM\_SU Constants <IN> 47](#_Toc21376067)

[Table 21 — Definition of (UINT8) TPM\_SE Constants <IN> 47](#_Toc21376068)

[Table 22 — Definition of (UINT32) TPM\_CAP Constants 48](#_Toc21376069)

[Table 23 — Definition of (UINT32) TPM\_PT Constants <IN/OUT, S> 49](#_Toc21376070)

[Table 24 — Definition of (UINT32) TPM\_PT\_PCR Constants <IN/OUT, S> 54](#_Toc21376071)

[Table 25 — Definition of (UINT32) TPM\_PS Constants <OUT> 56](#_Toc21376072)

[Table 26 — Definition of Types for Handles 57](#_Toc21376073)

[Table 27 — Definition of (UINT8) TPM\_HT Constants <S> 57](#_Toc21376074)

[Table 28 — Definition of (TPM\_HANDLE) TPM\_RH Constants <S> 59](#_Toc21376075)

[Table 29 — Definition of (TPM\_HANDLE) TPM\_HC Constants <S> 61](#_Toc21376076)

[Table 30 — Definition of (UINT32) TPMA\_ALGORITHM Bits 63](#_Toc21376077)

[Table 31 — Definition of (UINT32) TPMA\_OBJECT Bits 64](#_Toc21376078)

[Table 32 — Definition of (UINT8) TPMA\_SESSION Bits <IN/OUT> 71](#_Toc21376079)

[Table 33 — Definition of (UINT8) TPMA\_LOCALITY Bits <IN/OUT> 73](#_Toc21376080)

[Table 34 — Definition of (UINT32) TPMA\_PERMANENT Bits <OUT> 74](#_Toc21376081)

[Table 35 — Definition of (UINT32) TPMA\_STARTUP\_CLEAR Bits <OUT> 75](#_Toc21376082)

[Table 36 — Definition of (UINT32) TPMA\_MEMORY Bits <Out> 76](#_Toc21376083)

[Table 37 — Definition of (TPM\_CC) TPMA\_CC Bits <OUT> 77](#_Toc21376084)

[Table 38 — Definition of (UINT32) TPMA\_MODES Bits <Out> 79](#_Toc21376085)

[Table 39 — Definition of (UINT32) TPMA\_X509\_KEY\_USAGE Bits<> 80](#_Toc21376086)

[Table 40 — Definition of (UINT32) TPMA\_ACT Bits 81](#_Toc21376087)

[Table 41 — Definition of (BYTE) TPMI\_YES\_NO Type 82](#_Toc21376088)

[Table 42 — Definition of (TPM\_HANDLE) TPMI\_DH\_OBJECT Type 82](#_Toc21376089)

[Table 43 — Definition of (TPM\_HANDLE) TPMI\_DH\_PARENT Type 83](#_Toc21376090)

[Table 44 — Definition of (TPM\_HANDLE) TPMI\_DH\_PERSISTENT Type 83](#_Toc21376091)

[Table 45 — Definition of (TPM\_HANDLE) TPMI\_DH\_ENTITY Type <IN> 84](#_Toc21376092)

[Table 46 — Definition of (TPM\_HANDLE) TPMI\_DH\_PCR Type <IN> 84](#_Toc21376093)

[Table 47 — Definition of (TPM\_HANDLE) TPMI\_SH\_AUTH\_SESSION Type <IN/OUT> 85](#_Toc21376094)

[Table 48 — Definition of (TPM\_HANDLE) TPMI\_SH\_HMAC Type <IN/OUT> 85](#_Toc21376095)

[Table 49 — Definition of (TPM\_HANDLE) TPMI\_SH\_POLICY Type <IN/OUT> 85](#_Toc21376096)

[Table 50 — Definition of (TPM\_HANDLE) TPMI\_DH\_CONTEXT Type 85](#_Toc21376097)

[Table 51 — Definition of (TPM\_HANDLE) TPMI\_DH\_SAVED Type 86](#_Toc21376098)

[Table 52 — Definition of (TPM\_HANDLE) TPMI\_RH\_HIERARCHY Type 86](#_Toc21376099)

[Table 53 — Definition of (TPM\_HANDLE) TPMI\_RH\_ENABLES Type 86](#_Toc21376100)

[Table 54 — Definition of (TPM\_HANDLE) TPMI\_RH\_HIERARCHY\_AUTH Type <IN> 87](#_Toc21376101)

[Table 55 — Definition of (TPM\_HANDLE) TPMI\_RH\_HIERARCHY\_POLICY Type <IN> 87](#_Toc21376102)

[Table 56 — Definition of (TPM\_HANDLE) TPMI\_RH\_PLATFORM Type <IN> 87](#_Toc21376103)

[Table 57 — Definition of (TPM\_HANDLE) TPMI\_RH\_OWNER Type <IN> 88](#_Toc21376104)

[Table 58 — Definition of (TPM\_HANDLE) TPMI\_RH\_ENDORSEMENT Type <IN> 88](#_Toc21376105)

[Table 59 — Definition of (TPM\_HANDLE) TPMI\_RH\_PROVISION Type <IN> 88](#_Toc21376106)

[Table 60 — Definition of (TPM\_HANDLE) TPMI\_RH\_CLEAR Type <IN> 89](#_Toc21376107)

[Table 61 — Definition of (TPM\_HANDLE) TPMI\_RH\_NV\_AUTH Type <IN> 89](#_Toc21376108)

[Table 62 — Definition of (TPM\_HANDLE) TPMI\_RH\_LOCKOUT Type <IN> 89](#_Toc21376109)

[Table 63 — Definition of (TPM\_HANDLE) TPMI\_RH\_NV\_INDEX Type <IN/OUT> 90](#_Toc21376110)

[Table 64 — Definition of (TPM\_HANDLE) TPMI\_RH\_AC Type <IN> 90](#_Toc21376111)

[Table 65 — Definition of (TPM\_HANDLE) TPMI\_RH\_ACT Type 91](#_Toc21376112)

[Table 66 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_HASH Type 91](#_Toc21376113)

[Table 67 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_ASYM Type 91](#_Toc21376114)

[Table 68 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_SYM Type 92](#_Toc21376115)

[Table 69 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_SYM\_OBJECT Type 92](#_Toc21376116)

[Table 70 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_SYM\_MODE Type 92](#_Toc21376117)

[Table 71 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_KDF Type 93](#_Toc21376118)

[Table 72 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_SIG\_SCHEME Type 93](#_Toc21376119)

[Table 73 — Definition of (TPM\_ALG\_ID){ECC} TPMI\_ECC\_KEY\_EXCHANGE Type 93](#_Toc21376120)

[Table 74 — Definition of (TPM\_ST) TPMI\_ST\_COMMAND\_TAG Type 94](#_Toc21376121)

[Table 75 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_MAC\_SCHEME Type 94](#_Toc21376122)

[Table 76 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_CIPHER\_MODE Type 94](#_Toc21376123)

[Table 77 — Definition of TPMS\_EMPTY Structure <IN/OUT> 95](#_Toc21376124)

[Table 78 — Definition of TPMS\_ALGORITHM\_DESCRIPTION Structure <OUT> 95](#_Toc21376125)

[Table 79 — Definition of TPMU\_HA Union <IN/OUT > 95](#_Toc21376126)

[Table 80 — Definition of TPMT\_HA Structure <IN/OUT> 96](#_Toc21376127)

[Table 81 — Definition of TPM2B\_DIGEST Structure 97](#_Toc21376128)

[Table 82 — Definition of TPM2B\_DATA Structure 97](#_Toc21376129)

[Table 83 — Definition of Types for TPM2B\_NONCE 97](#_Toc21376130)

[Table 84 — Definition of Types for TPM2B\_AUTH 97](#_Toc21376131)

[Table 85 — Definition of Types for TPM2B\_OPERAND 98](#_Toc21376132)

[Table 86 — Definition of TPM2B\_EVENT Structure 98](#_Toc21376133)

[Table 87 — Definition of TPM2B\_MAX\_BUFFER Structure 98](#_Toc21376134)

[Table 88 — Definition of TPM2B\_MAX\_NV\_BUFFER Structure 98](#_Toc21376135)

[Table 89 — Definition of TPM2B\_TIMEOUT Structure 99](#_Toc21376136)

[Table 90 — Definition of TPM2B\_IV Structure <IN/OUT> 99](#_Toc21376137)

[Table 91 — Definition of TPMU\_NAME Union <> 99](#_Toc21376138)

[Table 92 — Definition of TPM2B\_NAME Structure 100](#_Toc21376139)

[Table 93 — Definition of TPMS\_PCR\_SELECT Structure 101](#_Toc21376140)

[Table 94 — Definition of TPMS\_PCR\_SELECTION Structure 101](#_Toc21376141)

[Table 95 — Values for *proof* Used in Tickets 102](#_Toc21376142)

[Table 96 — General Format of a Ticket 102](#_Toc21376143)

[Table 97 — Definition of TPMT\_TK\_CREATION Structure 103](#_Toc21376144)

[Table 98 — Definition of TPMT\_TK\_VERIFIED Structure 103](#_Toc21376145)

[Table 99 — Definition of TPMT\_TK\_AUTH Structure 104](#_Toc21376146)

[Table 100 — Definition of TPMT\_TK\_HASHCHECK Structure 105](#_Toc21376147)

[Table 101 — Definition of TPMS\_ALG\_PROPERTY Structure <OUT> 105](#_Toc21376148)

[Table 102 — Definition of TPMS\_TAGGED\_PROPERTY Structure <OUT> 105](#_Toc21376149)

[Table 103 — Definition of TPMS\_TAGGED\_PCR\_SELECT Structure <OUT> 106](#_Toc21376150)

[Table 104 — Definition of TPMS\_TAGGED\_POLICY Structure <OUT> 106](#_Toc21376151)

[Table 105 — Definition of TPMS\_ACT\_DATA Structure <OUT> 106](#_Toc21376152)

[Table 106 — Definition of TPML\_CC Structure 107](#_Toc21376153)

[Table 107 — Definition of TPML\_CCA Structure <OUT> 107](#_Toc21376154)

[Table 108 — Definition of TPML\_ALG Structure 107](#_Toc21376155)

[Table 109 — Definition of TPML\_HANDLE Structure <OUT> 108](#_Toc21376156)

[Table 110 — Definition of TPML\_DIGEST Structure 108](#_Toc21376157)

[Table 111 — Definition of TPML\_DIGEST\_VALUES Structure 109](#_Toc21376158)

[Table 112 — Definition of TPML\_PCR\_SELECTION Structure 109](#_Toc21376159)

[Table 113 — Definition of TPML\_ALG\_PROPERTY Structure <OUT> 110](#_Toc21376160)

[Table 114 — Definition of TPML\_TAGGED\_TPM\_PROPERTY Structure <OUT> 110](#_Toc21376161)

[Table 115 — Definition of TPML\_TAGGED\_PCR\_PROPERTY Structure <OUT> 110](#_Toc21376162)

[Table 116 — Definition of {ECC} TPML\_ECC\_CURVE Structure <OUT> 110](#_Toc21376163)

[Table 117 — Definition of TPML\_TAGGED\_POLICY Structure <OUT> 111](#_Toc21376164)

[Table 118 — Definition of TPML\_ACT\_DATA Structure <OUT> 111](#_Toc21376165)

[Table 119 — Definition of TPMU\_CAPABILITIES Union <OUT> 112](#_Toc21376166)

[Table 120 — Definition of TPMS\_CAPABILITY\_DATA Structure <OUT> 112](#_Toc21376167)

[Table 121 — Definition of TPMS\_CLOCK\_INFO Structure 113](#_Toc21376168)

[Table 122 — Definition of TPMS\_TIME\_INFO Structure 114](#_Toc21376169)

[Table 123 — Definition of TPMS\_TIME\_ATTEST\_INFO Structure <OUT> 115](#_Toc21376170)

[Table 124 — Definition of TPMS\_CERTIFY\_INFO Structure <OUT> 115](#_Toc21376171)

[Table 125 — Definition of TPMS\_QUOTE\_INFO Structure <OUT> 115](#_Toc21376172)

[Table 126 — Definition of TPMS\_COMMAND\_AUDIT\_INFO Structure <OUT> 116](#_Toc21376173)

[Table 127 — Definition of TPMS\_SESSION\_AUDIT\_INFO Structure <OUT> 116](#_Toc21376174)

[Table 128 — Definition of TPMS\_CREATION\_INFO Structure <OUT> 116](#_Toc21376175)

[Table 129 — Definition of TPMS\_NV\_CERTIFY\_INFO Structure <OUT> 116](#_Toc21376176)

[Table 130 — Definition of TPMS\_NV\_DIGEST\_CERTIFY\_INFO Structure <OUT> 117](#_Toc21376177)

[Table 131 — Definition of (TPM\_ST) TPMI\_ST\_ATTEST Type <OUT> 117](#_Toc21376178)

[Table 132 — Definition of TPMU\_ATTEST Union <OUT> 117](#_Toc21376179)

[Table 133 — Definition of TPMS\_ATTEST Structure <OUT> 118](#_Toc21376180)

[Table 134 — Definition of TPM2B\_ATTEST Structure <OUT> 119](#_Toc21376181)

[Table 135 — Definition of TPMS\_AUTH\_COMMAND Structure <IN> 119](#_Toc21376182)

[Table 136 — Definition of TPMS\_AUTH\_RESPONSE Structure <OUT> 119](#_Toc21376183)

[Table 137 — Definition of {!ALG.S} (TPM\_KEY\_BITS) TPMI\_!ALG.S\_KEY\_BITS Type 120](#_Toc21376184)

[Table 138 — Definition of TPMU\_SYM\_KEY\_BITS Union 120](#_Toc21376185)

[Table 139 — Definition of TPMU\_SYM\_MODE Union 121](#_Toc21376186)

[Table 140 —xDefinition of TPMU\_SYM\_DETAILS Union 121](#_Toc21376187)

[Table 141 — Definition of TPMT\_SYM\_DEF Structure 122](#_Toc21376188)

[Table 142 — Definition of TPMT\_SYM\_DEF\_OBJECT Structure 122](#_Toc21376189)

[Table 143 — Definition of TPM2B\_SYM\_KEY Structure 123](#_Toc21376190)

[Table 144 — Definition of TPMS\_SYMCIPHER\_PARMS Structure 123](#_Toc21376191)

[Table 145 — Definition of TPM2B\_LABEL Structure 123](#_Toc21376192)

[Table 146 — Definition of TPMS\_DERIVE Structure 123](#_Toc21376193)

[Table 147 — Definition of TPM2B\_DERIVE Structure 124](#_Toc21376194)

[Table 148 — Definition of TPMU\_SENSITIVE\_CREATE Union <> 124](#_Toc21376195)

[Table 149 — Definition of TPM2B\_SENSITIVE\_DATA Structure 124](#_Toc21376196)

[Table 150 — Definition of TPMS\_SENSITIVE\_CREATE Structure <IN> 125](#_Toc21376197)

[Table 151 — Definition of TPM2B\_SENSITIVE\_CREATE Structure <IN, S> 125](#_Toc21376198)

[Table 152 — Definition of TPMS\_SCHEME\_HASH Structure 125](#_Toc21376199)

[Table 153 — Definition of {ECC} TPMS\_SCHEME\_ECDAA Structure 126](#_Toc21376200)

[Table 154 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_KEYEDHASH\_SCHEME Type 126](#_Toc21376201)

[Table 155 — Definition of Types for HMAC\_SIG\_SCHEME 126](#_Toc21376202)

[Table 156 — Definition of TPMS\_SCHEME\_XOR Structure 126](#_Toc21376203)

[Table 157 — Definition of TPMU\_SCHEME\_KEYEDHASH Union <IN/OUT > 127](#_Toc21376204)

[Table 158 — Definition of TPMT\_KEYEDHASH\_SCHEME Structure 127](#_Toc21376205)

[Table 159 — Definition of {RSA} Types for RSA Signature Schemes 128](#_Toc21376206)

[Table 160 — Definition of {ECC} Types for ECC Signature Schemes 128](#_Toc21376207)

[Table 161 — Definition of TPMU\_SIG\_SCHEME Union <IN/OUT > 129](#_Toc21376208)

[Table 162 — Definition of TPMT\_SIG\_SCHEME Structure 129](#_Toc21376209)

[Table 163 — Definition of Types for {RSA} Encryption Schemes 129](#_Toc21376210)

[Table 164 — Definition of Types for {ECC} ECC Key Exchange 130](#_Toc21376211)

[Table 165 — Definition of Types for KDF Schemes 130](#_Toc21376212)

[Table 166 — Definition of TPMU\_KDF\_SCHEME Union <IN/OUT> 130](#_Toc21376213)

[Table 167 — Definition of TPMT\_KDF\_SCHEME Structure 130](#_Toc21376214)

[Table 168 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_ASYM\_SCHEME Type <IO> 131](#_Toc21376215)

[Table 169 — Definition of TPMU\_ASYM\_SCHEME Union 131](#_Toc21376216)

[Table 170 — Definition of TPMT\_ASYM\_SCHEME Structure <> 132](#_Toc21376217)

[Table 171 — Definition of (TPM\_ALG\_ID) {RSA} TPMI\_ALG\_RSA\_SCHEME Type 132](#_Toc21376218)

[Table 172 — Definition of {RSA} TPMT\_RSA\_SCHEME Structure 132](#_Toc21376219)

[Table 173 — Definition of (TPM\_ALG\_ID) {RSA} TPMI\_ALG\_RSA\_DECRYPT Type 132](#_Toc21376220)

[Table 174 — Definition of {RSA} TPMT\_RSA\_DECRYPT Structure 132](#_Toc21376221)

[Table 175 — Definition of {RSA} TPM2B\_PUBLIC\_KEY\_RSA Structure 133](#_Toc21376222)

[Table 176 — Definition of {RSA} (TPM\_KEY\_BITS) TPMI\_RSA\_KEY\_BITS Type 133](#_Toc21376223)

[Table 177 — Definition of {RSA} TPM2B\_PRIVATE\_KEY\_RSA Structure 133](#_Toc21376224)

[Table 178 — Definition of TPM2B\_ECC\_PARAMETER Structure 134](#_Toc21376225)

[Table 179 — Definition of {ECC} TPMS\_ECC\_POINT Structure 134](#_Toc21376226)

[Table 180 — Definition of {ECC} TPM2B\_ECC\_POINT Structure 134](#_Toc21376227)

[Table 181 — Definition of (TPM\_ALG\_ID) {ECC} TPMI\_ALG\_ECC\_SCHEME Type 135](#_Toc21376228)

[Table 182 — Definition of {ECC} (TPM\_ECC\_CURVE) TPMI\_ECC\_CURVE Type 135](#_Toc21376229)

[Table 183 — Definition of (TPMT\_SIG\_SCHEME) {ECC} TPMT\_ECC\_SCHEME Structure 135](#_Toc21376230)

[Table 184 — Definition of {ECC} TPMS\_ALGORITHM\_DETAIL\_ECC Structure <OUT> 136](#_Toc21376231)

[Table 185 — Definition of {RSA} TPMS\_SIGNATURE\_RSA Structure 136](#_Toc21376232)

[Table 186 — Definition of Types for {RSA} Signature 136](#_Toc21376233)

[Table 187 — Definition of {ECC} TPMS\_SIGNATURE\_ECC Structure 137](#_Toc21376234)

[Table 188 — Definition of Types for {ECC} TPMS\_SIGNATURE\_ECC 137](#_Toc21376235)

[Table 189 — Definition of TPMU\_SIGNATURE Union <IN/OUT> 137](#_Toc21376236)

[Table 190 — Definition of TPMT\_SIGNATURE Structure 137](#_Toc21376237)

[Table 191 — Definition of TPMU\_ENCRYPTED\_SECRET Union 138](#_Toc21376238)

[Table 192 — Definition of TPM2B\_ENCRYPTED\_SECRET Structure 138](#_Toc21376239)

[Table 193 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_PUBLIC Type 139](#_Toc21376240)

[Table 194 — Definition of TPMU\_PUBLIC\_ID Union <IN/OUT> 140](#_Toc21376241)

[Table 195 — Definition of TPMS\_KEYEDHASH\_PARMS Structure 141](#_Toc21376242)

[Table 196 — Definition of TPMS\_ASYM\_PARMS Structure <> 141](#_Toc21376243)

[Table 197 — Definition of {RSA} TPMS\_RSA\_PARMS Structure 142](#_Toc21376244)

[Table 198 — Definition of {ECC} TPMS\_ECC\_PARMS Structure 143](#_Toc21376245)

[Table 199 — Definition of TPMU\_PUBLIC\_PARMS Union <IN/OUT> 143](#_Toc21376246)

[Table 200 — Definition of TPMT\_PUBLIC\_PARMS Structure 144](#_Toc21376247)

[Table 201 — Definition of TPMT\_PUBLIC Structure 144](#_Toc21376248)

[Table 202 — Definition of TPM2B\_PUBLIC Structure 144](#_Toc21376249)

[Table 203 — Definition of TPM2B\_TEMPLATE Structure 145](#_Toc21376250)

[Table 204 — Definition of TPM2B\_PRIVATE\_VENDOR\_SPECIFIC Structure 145](#_Toc21376251)

[Table 205 — Definition of TPMU\_SENSITIVE\_COMPOSITE Union <IN/OUT> 146](#_Toc21376252)

[Table 206 — Definition of TPMT\_SENSITIVE Structure 146](#_Toc21376253)

[Table 207 — Definition of TPM2B\_SENSITIVE Structure <IN/OUT> 146](#_Toc21376254)

[Table 208 — Definition of \_PRIVATE Structure <> 147](#_Toc21376255)

[Table 209 — Definition of TPM2B\_PRIVATE Structure <IN/OUT> 147](#_Toc21376256)

[Table 210 — Definition of TPMS\_ID\_OBJECT Structure <> 148](#_Toc21376257)

[Table 211 — Definition of TPM2B\_ID\_OBJECT Structure <IN/OUT> 148](#_Toc21376258)

[Table 212 — Definition of (UINT32) TPM\_NV\_INDEX Bits <> 149](#_Toc21376259)

[Table 213 — Definition of TPM\_NT Constants 150](#_Toc21376260)

[Table 214 — Definition of TPMS\_NV\_PIN\_COUNTER\_PARAMETERS Structure 150](#_Toc21376261)

[Table 215 — Definition of (UINT32) TPMA\_NV Bits 152](#_Toc21376262)

[Table 216 — Definition of TPMS\_NV\_PUBLIC Structure 154](#_Toc21376263)

[Table 217 — Definition of TPM2B\_NV\_PUBLIC Structure 154](#_Toc21376264)

[Table 218 — Definition of TPM2B\_CONTEXT\_SENSITIVE Structure <IN/OUT> 155](#_Toc21376265)

[Table 219 — Definition of TPMS\_CONTEXT\_DATA Structure <IN/OUT> 155](#_Toc21376266)

[Table 220 — Definition of TPM2B\_CONTEXT\_DATA Structure <IN/OUT> 155](#_Toc21376267)

[Table 221 — Definition of TPMS\_CONTEXT Structure 156](#_Toc21376268)

[Table 222 — Context Handle Values 157](#_Toc21376269)

[Table 223 — Definition of TPMS\_CREATION\_DATA Structure <OUT> 159](#_Toc21376270)

[Table 224 — Definition of TPM2B\_CREATION\_DATA Structure <OUT> 159](#_Toc21376271)

[Table 225 — Definition of (UINT32) TPM\_AT Constants 160](#_Toc21376272)

[Table 226 — Definition of (UINT32) TPM\_AE Constants <OUT> 160](#_Toc21376273)

[Table 227 — Definition of TPMS\_AC\_OUTPUT Structure <OUT> 160](#_Toc21376274)

[Table 228 — Definition of TPML\_AC\_CAPABILITIES Structure <OUT> 161](#_Toc21376275)

Figures

[Figure 1 — Command Format 30](#_Toc20318083)

[Figure 2 — Format-Zero Response Codes 36](#_Toc20318084)

[Figure 3 — Format-One Response Codes 36](#_Toc20318085)

[Figure 4 — TPM 1.2 TPM\_NV\_INDEX 148](#_Toc20318086)

[Figure 5 — TPM 2.0 TPM\_NV\_INDEX 148](#_Toc20318087)

Trusted Platform Module Library

Part 2: Structures

# Scope

This part of the *Trusted Platform Module Library* specification contains the definitions of the constants, flags, structure, and union definitions used to communicate with the TPM. Values defined in this document are used by the TPM commands defined in TPM 2.0 Part 3: *Commands* and by the functions in TPM 2.0 Part 4: *Supporting Routines*.

NOTE The structures in this document are the canonical form of the structures on the interface. All structures are "packed" with no octets of padding between structure elements. The TPM-internal form of the structures is dependent on the processor and compiler for the TPM implementation.

# Terms and definitions

For the purposes of this document, the terms and definitions given in TPM 2.0 Part 1 apply.

# Symbols and abbreviated terms

For the purposes of this document, the symbols and abbreviated terms given in TPM 2.0 Part 1 apply.

# Notation

## Introduction

The information in this document is formatted so that it may be converted to standard computer-language formats by an automated process. The purpose of this automated process is to minimize the transcription errors that often occur during the conversion process.

For the purposes of this document, the conventions given in TPM 2.0 Part 1 apply.

In addition, the conventions and notations in clause 4 describe the representation of various data so that it is both human readable and amenable to automated processing.

When a table row contains the keyword “reserved” (all lower case) in columns 1 or 2, the tools will not produce any values for the row in the table.

NOTE The unmarshaling code examples are the actual code that would be produced by the automatic code generator used in the construction of the reference code. The actual code contains additional parameter checking that is omitted for clarity of the principle being illustrated. Actual examples of the code are found in TPM 2.0 Part 4.

## Named Constants

A named constant is a numeric value to which a name has been assigned. In the C language, this is done with a #define statement. In this specification, a named constant is defined in a table that has a title that starts with “Definition” and ends with “Constants.”

The table title will indicate the name of the class of constants that are being defined in the table. When applicable, the title will include the data type of the constants in parentheses.

The table in Example 1 names a collection of 16-bit constants and Example 2 shows the C code that might be produced from that table by an automated process.

NOTE A named constant (**#define**) has no data type in C and an enumeration would be a better choice for many of the defined constants. However, the C language does not allow an enumerated type to have a storage type other than **int** so the method of using a combination of **typedef** and **#define** is used.

EXAMPLE 1

Table xx — Definition of (UINT16) COUNTING Constants

|  |  |  |
| --- | --- | --- |
| Parameter | Value | Description |
| first | 1 | decimal value is implicitly the size of the |
| second | 0x0002 | hex value will match the number of bits in the constant |
| third | 3 |  |
| fourth | 0x0004 |  |

EXAMPLE 2

/\* The C language equivalent of the constants from the table above \*/

typedef UINT16 COUNTING;

#define first 1

#define second 0x0002

#define third 3

#define fourth 0x0004

## Data Type Aliases (typedefs)

When a group of named items is assigned a type, it is placed in a table that has a title starting with “Definition of Types.” In this specification, defined types have names that use all upper-case characters.

The table in Example 1 shows how typedefs would be defined in this specification and Example 2 shows the C-compatible code that might be produced from that table by an automated process.

EXAMPLE 1

Table xx — Definition of Types for Some Purpose

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| unsigned short | UINT16 |  |
| UINT16 | SOME\_TYPE |  |
| unsigned long | UINT32 |  |
| UINT32 | LAST\_TYPE |  |

EXAMPLE 2

/\* C language equivalent of the typedefs from the table above \*/

typedef unsigned short UINT16;

typedef UINT16 SOME\_TYPE;

typedef unsigned long UINT32;

typedef UINT32 LAST\_TYPE;

## Enumerations

A table that defines an enumerated data type will start with the word “Definition” and end with “Values.”

A value in parenthesis will denote the intrinsic data size of the value and may have the values "INT8", "UINT8", "INT16", “UINT16”, "INT32", and “UINT32.” If this value is not present, “UINT16” is assumed.

Most C compilers set the type of an enumerated value to be an integer on the machine – often 16 bits – but this is not always consistent. To ensure interoperability, the enumeration values may not exceed 32,384.

The table in Example 1 shows how an enumeration would be defined in this specification. Example 2 shows the C code that might be produced from that table by an automated process.

EXAMPLE 1

Table xx — Definition of (UINT16) CARD\_SUIT Values

|  |  |  |
| --- | --- | --- |
| Suit Names | Value | Description |
| CLUBS | 0x0000 |  |
| DIAMONDS | 0x000D |  |
| HEARTS | 0x001A |  |
| SPADES | 0x0027 |  |

EXAMPLE 2

/\* C language equivalent of the structure defined in the table above \*/

typedef enum {

CLUBS = 0x0000,

DIAMONDS = 0x000D,

HEARTS = 0x001A,

SPADES = 0x0027

} CARD\_SUIT;

## Interface Type

An interface type is used for an enumeration that is checked by the unmarshaling code. This type is defined for purposes of automatic generation of the code that will validate the type. The title will start with the keyword “Definition” and end with the keyword “Type.” A value in parenthesis indicates the base type of the interface. The table may contain an entry that is prefixed with the “#” character to indicate the response code if the validation code determines that the input parameter is the wrong type.

EXAMPLE 1

Table xx — Definition of (CARD\_SUIT) RED\_SUIT Type

| Values | Comments |
| --- | --- |
| HEARTS |  |
| DIAMONDS |  |
| #TPM\_RC\_SUIT | response code returned when the unmarshaling of this type fails  NOTE TPM\_RC\_SUIT is an example and no such response code is actually defined in this specification. |

EXAMPLE 2

/\* Validation code that might be automatically generated from table above \*/

if((\*target != HEARTS) && (\*target != DIAMONDS))

return TPM\_RC\_SUIT;

In some cases, the allowed values are numeric values with no associated mnemonic. In such a case, the list of numeric values may be given a name. Then, when used in an interface definition, the name would have a "$" prefix to indicate that a named list of values should be substituted.

To illustrate, assume that the implementation only supports two sizes (1024 and 2048 bits) for keys associated with some algorithm (MY algorithm).

EXAMPLE 3

Table xx — Defines for MY Algorithm Constants

|  |  |  |
| --- | --- | --- |
| Name | Value | Comments |
| MY\_KEY\_SIZES\_BITS | {1024, 2048} | braces because this is a list value |

Then, whenever an input value would need to be a valid MY key size for the implementation, the value $MY\_KEY\_SIZES\_BITS could be used. Given the definition for MY\_KEY\_SIZES\_BITS in example 3 above, the tables in example 4 and 5 below, are equivalent.

EXAMPLE 4

Table xx — Definition of (UINT16) MY\_KEY\_BITS Type

|  |  |
| --- | --- |
| Parameter | Description |
| {1024, 2048} | the number of bits in the supported key |

EXAMPLE 5

Table xx — Definition of (UINT16) MY\_KEY\_BITS Type

|  |  |
| --- | --- |
| Parameter | Description |
| $MY\_KEY\_SIZES\_BITS | the number of bits in the supported key |

## Arrays

Arrays are denoted by a value in square brackets (“[ ]”) following a parameter name. The value in the brackets may be either an integer value such as “[20]” or the name of a component of the same structure that contains the array.

The table in Example 1 shows how a structure containing fixed and variable-length arrays would be defined in this specification. Example 2 shows the C code that might be produced from that table by an automated process.

EXAMPLE 1

Table xx — Definition of A\_STRUCT Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| array1[20] | UINT16 | an array of 20 UINT16s |
| a\_size | UINT16 |  |
| array2[a\_size] | UINT32 | an array of UINT32 values that has a number of elements determined by a\_size above |

EXAMPLE 2

/\* C language equivalent of the typedefs from the table above \*/

typedef struct {

UINT16 array1[20];

UINT16 a\_size;

UINT32 array2[];

} A\_STRUCT;

## Structure Definitions

The tables used to define structures have a title that starts with the word “Definition” and ends with “Structure.” The first column of the table will denote the reference names for the structure members; the second column the data type of the member; and the third column a synopsis of the use of the element.

The table in Example 1 shows an example of how a structure would be defined in this specification and Example 2 shows the C code that might be produced from the table by an automated process. Example 3 illustrates the type of unmarshaling code that could be generated using the information available in the table.

EXAMPLE 1

Table xx — Definition of SIMPLE\_STRUCTURE Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| tag | TPM\_ST |  |
| value1 | INT32 |  |
| value2 | INT32 |  |

EXAMPLE 2

/\* C language equivalent of the structure defined in the table above \*/

typedef struct {

TPM\_ST tag;

INT32 value1

INT32 value2;

} SIMPLE\_STRUCTURE;

EXAMPLE 3

TPM\_RC SIMPLE\_STRUCTURE\_Unmarshal(SIMPLE\_STRUCTURE \*target, BYTE \*\*buffer, INT32 \*size)

{

TPM\_RC rc;

// If unmarshal of tag succeeds

rc = TPM\_ST\_Unmarshal((TPM\_ST \*)&(target->tag), buffer, size

If(rc == TPM\_RC\_SUCCESS)

{

// then unmarshal value1,

rc = INT32\_Unmarshal((INT32 \*)&(target->value1, buffer, size);

// and if that succeeds...

if(rc == TPM\_RC\_SUCCESS)

{

// then unmarshal the value 2

rc = INT32\_Unmarshal((INT32 \*)&(target->value2, buffer, size);

}

}

return rc;

}

A table may have a term in {}. This indicates that the table is conditionally compiled. It is commonly used when a table's inclusion is based on the implementation of a cryptographic algorithm. See, for example, Table 173 — Definition of (TPM\_ALG\_ID) {RSA} TPMI\_ALG\_RSA\_DECRYPT Type, which is dependent on the RSA algorithm.

## Conditional Types

An interface type may have a conditional value. This value is indicated by a “+” prepended to the name of the value. When this type is referenced in a structure, a “+” appended to the reference indicates that the instance allows the conditional value to be returned. If the reference does not has an appended “+”, then the conditional type is not allowed.

EXAMPLE 1 Table 66 defining TPMI\_ALG\_HASH indicates that TPM\_ALG\_NULL is a conditional type. TPMI\_ALG\_HASH is a member of the TPMS\_SCHEME\_XOR structure and that reference is TPMI\_ALG\_HASH+, indicating that TPM\_ALG\_NULL is an allowed value for hashAlg. TPMI\_ALG\_HASH is also referenced in TPMS\_PCR\_SELECTION. In that structure the TPMI\_ALG\_HASH does not have an appended “+”, so TPM\_ALG\_NULL would not be an allowed value for hash.

NOTE In many cases, the input values are algorithm IDs. When two collections of algorithm IDs differ only because one collection allows TPM\_ALG\_NULL and the other does not, it is preferred that there not be two completely different enumerations because this leads to many casts. To avoid this, the “+” can be added to a TPM\_ALG\_NULL value in the table defining the type. When the use of that type allows TPM\_ALG\_NULL to be in the set, the use would append a “+” to the instance.

When a type with a conditional value is referenced within a structure or union and the type reference has a “+” prepended to the type, it allows the references to that structure to treat it as if it had a conditional type. That means that a reference to that structure may have a “+” appended to the type. When the “+” is present in the structure/union reference, then the conditional value of the conditional type within the structure/union is allowed.

EXAMPLE 2 Table 142 — Definition of TPMT\_SYM\_DEF\_OBJECT Structure defines the TPMT\_SYM\_DEF\_OBJECT. The algorithm element of that structure is a TPMI\_ALG\_SYM\_OBJECT with a “+” prepended. This means that when a TPMT\_SYM\_DEF\_OBJECT is referenced, the reference may have an appended “+” as it does in the definition of the symmetric parameter of TPMS\_ASYM\_PARMS. The “+” in TPMA\_ASYM\_PARMS means that the algorithm parameter in the TPMT\_SYM\_DEF\_OBJECT may have the conditional value (TPM\_ALG\_NULL).

EXAMPLE 3

Table xx — Definition of (CARD\_SUIT) TPMI\_CARD\_SUIT Type

| Values | Comments |
| --- | --- |
| SPADES |  |
| HEARTS |  |
| DIAMONDS |  |
| CLUBS |  |
| +JOKER | an optional value that may be allowed |
| #TPM\_RC\_SUIT | response code returned when the input value is not one of the values above |

EXAMPLE 4

Table xx — Definition of POKER\_CARD Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| suit | TPMI\_CARD\_SUIT+ | allows joker |
| number | UINT8 | the card value |

EXAMPLE 5

Table xx — Definition of BRIDGE\_CARD Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| suit | TPMI\_CARD\_SUIT | does not allow joker |
| number | UINT8 | the card value |

## Unions

### Introduction

A union allows a structure to contain a variety of structures or types. The union has members, only one of which is present at a time. Three different tables are required to fully characterize a union so that it may be communicated on the TPM interface and used by the TPM:

* union definition;
* union instance; and
* union selector definition.

### Union Definition

The table in Example 1 illustrates a union definition. The title of a union definition table starts with “Definition” and ends with “Union.” The “Parameter” column of a union definition lists the different names that are used when referring to a specific type. The “Type” column identifies the data type of the member. The “Selector” column identifies the value that is used by the marshaling and unmarshaling code to determine which case of the union is present.

If a parameter is the keyword “null” or the type is empty, then this denotes a selector with no contents. The table in Example 1 illustrates a union in which a conditional null selector is allowed to indicate an empty union member.

Example 2 shows how the table would be converted into C-compatible code.

The expectation is that the unmarshaling code for the union will validate that the selector for the union is one of values in the selector list.

EXAMPLE 1

Table xx — Definition of NUMBER\_UNION Union

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| a\_byte | BYTE | BYTE\_SELECT |  |
| an\_int | int | INT\_SELECT |  |
| a\_float | float | FLOAT\_SELECT |  |
| +null |  | NULL\_SELECT | the empty branch |

EXAMPLE 2

// C-compatible version of the union defined in the table above

typedef union {

BYTE a\_byte;

int an\_int;

float a\_float;

} NUMBER\_UNION;

EXAMPLE 3

// Possible auto-generated code to unmarshal a union in Example 2 based on the

// input value of selector

TPM\_RC NUMBER\_UNION\_Unmarshal(NUMBER\_UNION \*target, BYTE \*\*buffer,

INT32 \*size, UINT32 selector)

{

switch (selector) {

case BYTE\_SELECT:

return BYTE\_Unmarshal((BYTE \*)&(target->a\_byte), buffer, size);

case INT\_SELECT:

return INT\_Unmarshal((int \*)&(target->an\_int), buffer, size);

case FLOAT\_SELECT:

return FLOAT\_Unmarshal((float \*)&(target->a\_float), buffer, size);

case NULL\_SELECT:

return TPM\_RC\_SUCCESS;

}

A table may have a type with no selector. This is used when the first part of the structure for all union members is identical. This type is a programming convenience, allowing code to reference the common members without requiring a case statement to determine the specific structure. In object oriented programming terms, this type is a superclass and the types with selectors are subclasses. Since there is no selector, this union member cannot be marshaled or unmarshaled.

EXAMPLE 4 Table 189 has an 'any' parameter with no selector. Any of the other union members may be cast to TPMS\_SCHEME\_HASH, since all begin with TPMI\_ALG\_HASH.

### Union Instance

When a union is used in a structure that is sent on the interface, the structure will minimally contain a selector and a union. The selector value indicates which of the possible union members is present so that the unmarshaling code can unmarshal the correct type. The selector may be any of the parameters that occur in the structure before the union instance. To denote the structure parameter that is used as the selector, its name is in brackets (“[ ]”) placed before the parameter name associated with the union.

The table in Example 1 shows the definition of a structure that contains a union and a selector. Example 2 shows how the table would be converted into C-compatible code and Example 3 shows how the unmarshaling code would handle the selector.

EXAMPLE 1

Table xx — Definition of STRUCTURE\_WITH\_UNION Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| select | NUMBER\_SELECT | a value indicating the type in *number* |
| [select] number | NUMBER\_UNION | a union as shown in 4.9.2 |

EXAMPLE 2

// C-compatible version of the union structure in the table above

typedef struct {

NUMBER\_SELECT select;

NUMBER\_UNION number;

} STRUCT\_WITH\_UNION;

EXAMPLE 3

// Possible unmarshaling code for the structure above

TPM\_RC STRUCT\_WITH\_UNION\_Unmarshal(STRUCT\_WITH\_UNION \*target, BYTE \*\*buffer, INT32 \*size)

{

TPM\_RC rc;

// Unmarshal the selector value

rc = NUMBER\_SELECT\_Unmarshal((NUMBER\_SELECT \*)&target->select, buffer, size)

if(rc != TPM\_RC\_SUCCESS)

return rc;

// Use the unmarshaled selector value to indicate to the union unmarshal

// function which unmarshaling branch to follow.

return(NUMBER\_UNION\_Unmarshal((NUMBER\_UNION \*)&(target->number),

buffer, size, (UINT32)target->select);

}

### Union Selector Definition

The selector definition limits the values that are used in unmarshaling a union. Two different selector sets applied to the same union define different types.

For the union in 4.9.2, a selector definition should be limited to no more than four values, one for each of the union members. The selector definition could have fewer than four values.

In Example 1, the table defines a value for each of the union members.

EXAMPLE 1

Table xx — Definition of (INT8) NUMBER\_SELECT Values <IN>

|  |  |  |
| --- | --- | --- |
| Name | Value | Comments |
| BYTE\_SELECT | 3 |  |
| INT\_SELECT | 2 |  |
| FLOAT\_SELECT | 1 |  |
| NULL\_SELECT | 0 |  |

The unmarshaling code would limit the input values to the defined values. When the NUMBER\_SELECT is used in the union instance of 4.9.3, any of the allowed union members of NUMBER\_UNION could be present.

A different selection could be used to limit the values in a specific instance. To get the different selection, a new structure is defined with a different selector. The table in example 2 illustrates a way to subset the union. The base type of the selection is NUMBER\_SELECT so a NUMBER\_SELECT will be unmarshaled before the checks are made to see if the value is in the correct range for JUST\_INTEGERS types. If the base type had been UINT8, then no checking would occur prior to checking that the value is in the allowed list. In this particular case, the effect is the same in either case since the only values that will be accepted by the unmarshaling code for JUST\_INTEGER are BYTE\_SELECT and INT\_SELECT.

EXAMPLE 2

Table xx — Definition of (NUMBER\_SELECT) AN\_INTEGER Type <IN>

| Values | Comments |
| --- | --- |
| {BYTE\_SELECT, INT\_SELECT} | list of allowed values |

NOTE Since NULL\_SELECT is not in the list of values accepted as a JUST\_INTEGER, the “+” modifier will have no effect if used for a JUST\_INTEGERS type shown in Example 3.

The selector in Example 2 can then be used in a subset union as shown in Example 3.

EXAMPLE 3

Table xx — Definition of JUST\_INTEGERS Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| select | AN\_INTEGER | a value indicating the type in *number* |
| [select] number | NUMBER\_UNION | a union as shown in 4.9.2 |

## Bit Field Definitions

A table that defines a structure containing bit fields has a title that starts with “Definition” and ends with “Bits.” A type identifier in parentheses in the title indicates the size of the datum that contains the bit fields.

When the bit fields do not occupy consecutive locations, a spacer field is defined with a name of “Reserved.” Bits in these spaces are reserved and shall be zero.

The table in Example 1 shows how a structure containing bit fields would be defined in this specification. Example 2 shows the C code that might be produced from that table by an automated process.

When a field has more than one bit, the range is indicated by a pair of numbers separated by a colon (“:”). The numbers will be in high:low order.

EXAMPLE1

Table xx — Definition of (UINT32) SOME\_ATTRIBUTE Bits

|  |  |  |
| --- | --- | --- |
| Bit | Name | Action |
| 0 | zeroth\_bit | **SET (1)**: what to do if bit is 1  **CLEAR (0):** what to do if bit is 0 |
| 1 | first\_bit | **SET (1)**: what to do if bit is 1  **CLEAR (0):** what to do if bit is 0 |
| 6:2 | Reserved | A placeholder that spans 5 bits |
| 7 | third\_bit | **SET (1)**: what to do if bit is 1  **CLEAR (0):** what to do if bit is 0 |
| 31:8 | Reserved | Placeholder to fill 32 bits |

EXAMPLE 2

/\* C language equivalent of the attributes structure defined in the table above \*/

typedef struct {

int zeroth\_bit : 1;

int first\_bit : 1;

int Reserved3 : 5;

int third\_bit : 1;

int Reserved7 : 24;

} SOME\_ATTRIBUTE;

NOTE The packing of bit fields into an integer is compiler and tool chain dependent. This C language equivalent is valid for a compiler that packs bit fields from the least significant bit to the most significant bit. It is likely to be correct for a little endian processor and likely to be incorrect for a big endian processor.

## Parameter Limits

A parameter used in a structure may be given a set of values that can be checked by the unmarshaling code. The allowed values for a parameter may be included in the definition of the parameter by appending the values and delimiting them with braces (“{ }”). The values are comma-separated expressions. A range of numbers may be indicated by separating two expressions with a colon (“:”). The first number is an expression that represents the minimum allowed value and the second number indicates the maximum. If the minimum or maximum value expression is omitted, then the range is open-ended.

Lower limits expressed using braces apply only to inputs to the TPM. The lower limit for a value returned by the TPM is determined by input parameters and the TPM implementation. Upper limits expressed using braces apply to both inputs to and outputs from the TPM.

NOTE In many cases, the upper limits are dependent on the TPM implementation. The values for these limits can be determined by accessing the TPM’s capabilities.

The maximum size of an array may be indicated by putting a “{}” delimited expression following the square brackets (“[ ]”) that indicate that the value is an array.

EXAMPLE

Table xx — Definition of B\_STRUCT Structure

| Parameter | Type | Description |
| --- | --- | --- |
| value1 {20:25} | UINT16 | a parameter that must have a value between 20 and 25, inclusive |
| value2 {20} | UINT16 | a parameter that must have a value of 20 |
| value3 {:25} | INT16 | a parameter that may be no larger than 25  Since the parameter is signed, the minimum value is the largest negative integer that may be expressed in 16 bits. |
| value4 {20:} |  | a parameter that must be at least 20 |
| value5 {1,2,3,5} | UINT16 | a parameter that may only have one of the four listed values |
| value6 {1, 2, 10:(10+10)} | UINT32 | a parameter that may have a value of 1, 2, or be between 10 and 20 |
| array1[value1] | BYTE | Because the index refers to *value1*, which is a value limited to be between 20 and 25 inclusive, array1 is an array that may have between 20 and 25 octets. This is not the preferred way to indicate the upper limit for an array as it does not indicate the upper bound of the size.  NOTE This is a limitation of the current parser. A different parser could associate the range of value1 with this value and compute the maximum size of the array. |
| array2[value4]{:25} | BYTE | an array that may have between 20 and 25 octets  This arrangement is used to allow the automatic code generation to allocate 25 octets to store the largest array2 that can be unmarshaled. The code generation can determine from this expression that *value4* shall have a value of 25 or less. From the definition of *value4* above, it can determine that *value4* must have a value of at least 20. |

## Algorithm Macros

### Introduction

This specification is intended to be algorithm agile in two different ways. In the first, agility is provided by allowing different subsets of the algorithms listed in the TCG registry. In the second, agility is provided by allowing the list of algorithms in the TCG registry to change without requiring changes to this specification.

This second form of algorithm agility is accomplished by using placeholder tokens that represent all of the algorithms of a particular type. The type of the algorithm is indicated by the letters in the Type column of the TPM\_ALG\_ID table in the TCG registry.

The use of these tokens is described in the remainder of this clause 4.12.

### Algorithm Token Semantics

The string “!ALG” or “!alg” indicates the algorithm token. This token may be followed by an algorithm type selection. The presence of the type selection is indicated by a period (“.”) following the token. The selection is all alphanumeric characters following the period.

NOTE In this selection context, the underscore character (“\_”) is not considered an alphanumeric character.

The selection is either an exclusive selection or an inclusive selection. An exclusive selection is one for which the Type entry for the algorithm is required to exactly match the type selection of the token. An inclusive selection is one where the Type entry for the algorithm is required to contain all of the characters of the selection but may contain additional attributes.

EXAMPLE 1 The “!ALG.AX” token would select those algorithms that only have the ‘A’ and ‘X’ types (that is, an asymmetric signing algorithm). The “!ALG.ax” token would select those algorithms that at least have ‘A’ and ‘X’ types but would include algorithms with other types such as ‘ANX’ (asymmetric signing and anonymous asymmetric signing).

When a replacement is made, the token will be replaced by an algorithm root identifier using either upper or lower case. If the algorithm token is part of another word, then the replacement uses upper case characters, otherwise, lower case is used.

NOTE The root identifier of an algorithm is the name in the TPM\_ALG\_ID table with “TPM\_ALG\_” removed. For example TPM\_ALG\_SHA1 has “SHA1” as its root.

The typical use of these tokens follows.

### Algorithm Tokens in Unions

A common place for algorithm tokens is in a union of values that are dependent on the type of the algorithm

EXAMPLE 1 An algorithm token indicating all hashes would be “!ALG.H” and could be used in a table to indicate that a union contains all defined hashes.

Table A — Definition of TPMU\_HA Union

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| !ALG.H [!ALG\_DIGEST\_SIZE] | BYTE | TPM\_ALG\_!ALG | all hashes |
| null |  | TPM\_ALG\_NULL |  |

If the TCG registry only contained SHA1, SHA256, and the SM3\_256 hash algorithm identifiers, then the table above would be semantically equivalent to:

Table xx — Definition of TPMU\_HA Union

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| sha1 [SHA1\_DIGEST\_SIZE] | BYTE | TPM\_ALG\_SHA1 |  |
| sha256 [SHA256\_DIGEST\_SIZE] | BYTE | TPM\_ALG\_SHA256 |  |
| sm3\_256 [SM3\_256\_DIGEST\_SIZE] | BYTE | TPM\_ALG\_SM3\_256 |  |
| null |  | TPM\_ALG\_NULL |  |

As shown in table A, the case of the replacement is determined by context. When !ALG is not an element of a longer name, then lower case characters are used. When !ALG is part of a longer name (indicated by leading or trailing underscore (“\_”), then upper case is used for the replacement.

Only one occurrence of the algorithm type (such as !ALG.H) is required for a line. If a line contains multiple list selections they are required to be identical.

If a table contains multiple lines with algorithm tokens, then each line is expanded separately.

### Algorithm Tokens in Interface Types

An interface type is often used with a union to create a tagged structure – the structure contains a union and a tag to indicate which of the union elements is actually present. The interface type for a tagged structure will usually contain the same elements as the union.

EXAMPLE If SHA1, SHA256, and SM3\_256 are the only defined hash algorithms, then an interface type to select a hash would be:

Table xx — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_HASH Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_SHA1 | example |
| TPM\_ALG\_SHA256 | example |
| TPM\_ALG\_SM3\_256 | example |
| +TPM\_ALG\_NULL |  |
| #TPM\_RC\_HASH |  |

An equivalent table may be represented using an algorithm macro.

Table xx — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_HASH Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.H | all hash algorithms defined by the TCG |
| +TPM\_ALG\_NULL |  |
| #TPM\_RC\_HASH |  |

### Algorithm Tokens for Table Replication

When a table is used to define an algorithm-specific value, that table may be replicated using the algorithm replacement token to create a table with values specific to the algorithm type. This type of replication is indicated by using an algorithm token in the name of the table.

EXAMPLE If AES and SM4 are the only defined symmetric block ciphers, then:

Table xx — Definition of {!ALG.S} (TPM\_KEY\_BITS) TPMI\_!ALG\_KEY\_BITS Type

|  |  |
| --- | --- |
| Parameter | Description |
| $!ALG\_KEY\_SIZES\_BITS | number of bits in the key |
| #TPM\_RC\_VALUE | error when key size is not supported |

has the same meaning as:

Table xx — Definition of {AES} (TPM\_KEY\_BITS) TPMI\_AES\_KEY\_BITS Type

|  |  |
| --- | --- |
| Parameter | Description |
| $AES\_KEY\_SIZES\_BITS | number of bits in the key |
| #TPM\_RC\_VALUE | error when key size is not supported |

Table xx — Definition of {SM4} (TPM\_KEY\_BITS) TPMI\_SM4\_KEY\_BITS Type

|  |  |
| --- | --- |
| Parameter | Description |
| $SM4\_KEY\_SIZES\_BITS | number of bits in the key |
| #TPM\_RC\_VALUE | error when key size is not supported |

## Size Checking

In some structures, a size field is present to indicate the number of octets in some subsequent part of the structure. In the B\_STRUCT table in 4.11, *value4* indicates how many octets to unmarshal for *array2*. This semantic applies when the size field determines the number of octets to unmarshal. However, in some cases, the subsequent structure is self-defining. If the size precedes a parameter that is not an octet array, then the unmarshaled size of that parameter is determined by its data type. The table in Example 1 shows a structure where the size parameter would nominally indicate the number of octets in the remainder of the structure.

EXAMPLE 1

Table xx — Definition of C\_STRUCT Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Comments |
| size | UINT16 | the expected size of the remainder of the structure |
| anInteger | UINT32 | a 4-octet value |

In this particular case, the value of size would be incorrect if it had any value other than 4. So that the table parser is able to know that the purpose of the size parameter is to define the number of octets expected in the remainder of the structure, an equal sign (“=”) is appended to the parameter name.

In the example below, the *size=* causes the parser to generate validation code that will check that the unmarshaled size of *someStructure* and *someData* adds to the value unmarshaled for *size*. When the “=” decoration is present, a value of zero is not allowed for the size.

EXAMPLE 2

Table xx — Definition of D\_STRUCT Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Comments |
| size= | UINT16 | the size of a structure  The “=” indicates that the TPM is required to validate that the remainder of the D\_STRUCT structure is exactly the value in *size*. That is, the number of bytes in the input buffer used to successfully unmarshal *someStructure* must be the same as *size*. |
| someStructure | A\_STRUCT | a structure to be unmarshaled  The size of the structure is computed when it is unmarshaled. Because an “=” is present on the definition of *size,* the TPM is required to validate that the unmarshaled size exactly matches *size*. |
| someData | UINT32 | a value |

## Data Direction

A structure or union may be input (IN), output (OUT), or internal. An input structure is sent to the TPM and is unmarshaled by the TPM. An output structure is sent from the TPM and is marshaled by the TPM. An internal structure is not used outside of the TPM except that it may be included in a saved context.

By default, structures are assumed to be both IN and OUT and the code generation tool will generate both marshaling and unmarshaling code for the structure. This default may be changed by using values enclosed in angle brackets (“<>”) as part of the table title. If the angle brackets are empty, then the structure is internal and neither marshaling nor unmarshaling code is generated. If the angle brackets contain the letter “I” (such as in “IN” or “in” or “i”), then the structure is input and unmarshaling code will be generated. If the angle brackets contain the letter “O” (such as in “OUT” or “out” or “o”), then the structure is output and marshaling code will be generated.

EXAMPLE 1 Both of the following table titles would indicate a structure that is used in both input and output

**Table xx — Definition of TPMS\_A Structure**

**Table xx — Definition of TPMS\_A Structure <IN/OUT>**

EXAMPLE 2 The following table title would indicate a structure that is used only for input

**Table xx — Definition of TPMS\_A Structure <IN>**

EXAMPLE 3 The following table title would indicate a structure that is used only for output

**Table xx — Definition of TPMS\_A Structure <OUT>**

## Structure Validations

By default, when a structure is used for input to the TPM, the code generation tool will generate the unmarshaling code for that structure. Auto-generation may be suppressed by adding an “S” within the angle brackets.

EXAMPLE The following table titles indicate a structure for which the auto-generation of the validation code is to be suppressed.

**Table xx — Definition of TPMT\_A Structure <S>**

**Table xx — Definition of TPMT\_A Structure <IN, S>**

**Table xx — Definition of TPMT\_A Structure <IN/OUT, S>**

## Name Prefix Convention

Parameters are constants, variables, structures, unions, and structure members. Structure members are given a name that is indicative of its use, with no special prefix. The other parameter types are named according to their type with their name starting with “TPMx\_”, where “x” is an optional character to indicate the data type.

In some cases, additional qualifying characters will follow the underscore. These are generally used when dealing with an enumerated data type.

Table 1 — Name Prefix Convention

| Prefix | Description |
| --- | --- |
| \_TPM\_ | an indication/signal from the TPM’s system interface |
| TPM\_ | a constant or an enumerated type |
| TPM2\_ | a command defined by this specification |
| TPM2B\_ | a structure that is a sized buffer where the size of the buffer is contained in a 16-bit, unsigned value  The first parameter is the size in octets of the second parameter. The second parameter may be any type. |
| TPMA\_ | a structure where each of the fields defines an attribute and each field is usually a single bit  All the attributes in an attribute structure are packed with the overall size of the structure indicated in the heading of the attribute description (UINT8, UINT16, or UINT32). |
| TPM\_ALG\_ | an enumerated type that indicates an algorithm  A TPM\_ALG\_ is often used as a selector for a union. |
| TPMI\_ | an interface type  The value is specified for purposes of dynamic type checking when unmarshaled. |
| TPML\_ | a list length followed by the indicated number of entries of the indicated type  This is an array with a length field. |
| TPMS\_ | a structure that is not a size buffer or a tagged buffer or a list |
| TPMT\_ | a structure with the first parameter being a structure tag, indicating the type of the structure that follows  A structure tag may be either a TPM\_ST\_ or TPM\_ALG\_ depending on context. |
| TPMU\_ | a union of structures, lists, or unions  If a union exists, there will normally be a companion TPMT\_ that is the expression of the union in a tagged structure, where the tag is the selector indicating which member of the union is present. |
| TPM\_xx\_ | an enumeration value of a particular type  The value of “xx” will be indicative of the use of the enumerated type. A table of “TPM\_xx” constant definitions will exist to define each of the TPM\_xx\_ values.  EXAMPLE 1 TPM\_CC\_ indicates that the type is used for a *commandCode.* The allowed enumeration values will be found in the table defining the TPM\_CC constants (Table 12)  EXAMPLE 2 TPM\_RC\_ indicates that the type is used for a *responseCode*. The allowed enumeration values are in Table 16. |

## Data Alignment

The data structures in this TPM 2.0 Part 2 use octet alignment for all structures. When used in a table to indicate a maximum size, the sizeof() function returns the octet-aligned size of the structure, with no padding.

## Parameter Unmarshaling Errors

The TPM commands are defined in TPM 2.0 Part 3. The command definition includes C code that details the actions performed by that command. The code is written assuming that the parameters of the command have been unmarshaled.

NOTE 1 An implementation is not required to process parameters in this manner or to separate the parameter parsing from the command actions. This method was chosen for the specification so that the normative behavior described by the detailed actions would be clear and unencumbered.

Unmarshaling is the process of processing the parameters in the input buffer and preparing the parameters for use by the command-specific action code. No data movement need take place but it is required that the TPM validate that the parameters meet the requirements of the expected data type as defined in this TPM 2.0 Part 2.

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.

EXAMPLE 1 Table 12 has a listing of TPM command code values. The last row in the table contains "#TPM\_RC\_COMMAND\_CODE" indicating the response code that is returned if the TPM is unmarshaling a value that it expects to be a TPM\_CC and the input value is not in the table.

NOTE 2 In the reference implementation, a parameter number is added to the response code so that the offending parameter can be isolated.

In many cases, the table contains no specific response code value and the return code will be determined as defined in Table 2.

Table 2 — Unmarshaling Errors

|  |  |
| --- | --- |
| Response code | Usage |
| TPM\_RC\_INSUFFICIENT | the input buffer did not contain enough octets to allow unmarshaling of the expected data type; |
| TPM\_RC\_RESERVED\_BITS | a non-zero value was found in a reserved field of an attribute structure (TPMA\_) |
| TPM\_RC\_SIZE | the value of a size parameter is larger or smaller than allowed |
| TPM\_RC\_VALUE | A parameter does not have one of its allowed values |
| TPM\_RC\_TAG | A parameter that should be a structure tag has a value that is not supported by the TPM |

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 TPM 2.0 Part 2 definition of the parameter type even if that parameter is not used in the command actions.

# Base Types

## Primitive Types

The types listed in Table 3 are the primitive types on which all of the other types and structures are based. The values in the “Type” column should be edited for the compiler and computer on which the TPM is implemented. The values in the “Name” column should remain the same because these values are used in the remainder of the specification.

NOTE The types are compatible with the C99 standard and should be defined in stdint.h that is provided with a C99-compliant compiler;

The parameters in the Name column should remain in the order shown.

Table 3 — Definition of Base Types

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| uint8\_t | UINT8 | unsigned, 8-bit integer |
| uint8\_t | BYTE | unsigned 8-bit integer |
| int8\_t | INT8 | signed, 8-bit integer |
| int | BOOL | a bit in an **int**  This is not used across the interface but is used in many places in the code. If the type were sent on the interface, it would have to have a type with a specific number of bytes. |
| uint16\_t | UINT16 | unsigned, 16-bit integer |
| int16\_t | INT16 | signed, 16-bit integer |
| uint32\_t | UINT32 | unsigned, 32-bit integer |
| int32\_t | INT32 | signed, 32-bit integer |
| uint64\_t | UINT64 | unsigned, 64-bit integer |
| int64\_t | INT64 | signed, 64-bit integer |

## Specification Logic Value Constants

Table 4 — Defines for Logic Values

|  |  |  |
| --- | --- | --- |
| Name | Value | Description |
| TRUE | 1 |  |
| FALSE | 0 |  |
| YES | 1 |  |
| NO | 0 |  |
| SET | 1 |  |
| CLEAR | 0 |  |

## Miscellaneous Types

These types are defined either for compatibility with previous versions of this specification or for clarity of this specification.

Table 5 — Definition of Types for Documentation Clarity

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| UINT32 | TPM\_ALGORITHM\_ID | this is the 1.2 compatible form of the TPM\_ALG\_ID |
| UINT32 | TPM\_MODIFIER\_INDICATOR |  |
| UINT32 | TPM\_AUTHORIZATION\_SIZE | the *authorizationSize* parameter in a command |
| UINT32 | TPM\_PARAMETER\_SIZE | the *parameterSize* parameter in a command |
| UINT16 | TPM\_KEY\_SIZE | a key size in octets |
| UINT16 | TPM\_KEY\_BITS | a key size in bits |

# Constants

## TPM\_SPEC (Specification Version Values)

These values are readable with TPM2\_GetCapability() (see 6.13 for the format).

NOTE 1 This table will require editing when the specification is updated.

NOTE 2 The year and day of year are those of this specification if the TPM does not implement errata. If the TPM implements errata, the values indicate the release date of the errata document. There is no provision for indicating that not all errata are implemented.

Table 6 — Definition of (UINT32) TPM\_SPEC Constants <>

| Name | Value | Comments |
| --- | --- | --- |
| TPM\_SPEC\_FAMILY | 0x322E3000 | ASCII “2.0” with null terminator |
| TPM\_SPEC\_LEVEL | 00 | the level number for the specification |
| TPM\_SPEC\_VERSION | 159 | the version number of the spec (001.59 \* 100) |
| TPM\_SPEC\_YEAR | 2019 | the year of the version |
| TPM\_SPEC\_DAY\_OF\_YEAR | 312 | the day of the year (November 8) |

## TPM\_GENERATED

This constant value differentiates TPM-generated structures from non-TPM structures.

Table 7 — Definition of (UINT32) TPM\_GENERATED Constants <O>

|  |  |  |
| --- | --- | --- |
| Name | Value | Comments |
| TPM\_GENERATED\_VALUE | 0xff544347 | 0xFF ‘TCG’ (FF 54 43 4716) |

## TPM\_ALG\_ID

The TCG maintains a registry of all algorithms that have an assigned algorithm ID. That registry is the definitive list of algorithms that may be supported by a TPM.

NOTE Inclusion of an algorithm does NOT indicate that the necessary claims of the algorithm are available under reasonable and non-discriminatory (RAND) terms from a TCG member.

Table 9 is an informative example of a TPM\_ALG\_ID constants table in the TCG Algorithm registry. Table 9 is provided for illustrative purposes only.

An algorithm ID is often used like a tag to determine the type of a structure in a context-sensitive way. The values for TPM\_ALG\_ID shall be in the range of 00 0016 – 7F FF16. Other structure tags will be in the range 80 0016 – FF FF16.

NOTE In TPM 1.2, these were defined as 32-bit constants. This specification limits the future size of the algorithm ID to 16 bits. The TPM\_ALGORITHM\_ID data type will continue to be a 32-bit number.

An algorithm shall not be assigned a value in the range 00 C116 – 00 C616 in order to prevent any overlap with the command structure tags used in TPM 1.2.

The implementation of some algorithms is dependent on the presence of other algorithms. When there is a dependency, the algorithm that is required is listed in column labeled "D" (dependent) in Table 9.

EXAMPLE Implementation of TPM\_ALG\_RSASSA requires that the RSA algorithm be implemented.

TPM\_ALG\_KEYEDHASH and TPM\_ALG\_NULL are required of all TPM implementations.

Table 8 — Legend for TPM\_ALG\_ID Table

| Column Title | Comments |
| --- | --- |
| Algorithm Name | the mnemonic name assigned to the algorithm |
| Value | the numeric value assigned to the algorithm |
| Type | The allowed values are:  **A** – asymmetric algorithm with a public and private key  **S** – symmetric algorithm with only a private key  **H** – hash algorithm that compresses input data to a digest value or indicates a method that uses a hash  **X** – signing algorithm  **N** – an anonymous signing algorithm  **E** – an encryption algorithm  **M** – a method such as a mask generation function  **O** – an object type |
| C | (**C**lassification) The allowed values are:  **A** – Assigned  **S** – TCG Standard  **L** – TCG Legacy |
| Dep | (**D**ependent) Indicates which other algorithm is required to be implemented if this algorithm is implemented |
| Reference | the reference document that defines the algorithm |
| Comments | clarifying information |

Table 9 — Definition of (UINT16) TPM\_ALG\_ID Constants <IN/OUT, S>

| Algorithm Name | Value | Type | Dep | C | Reference | Comments |
| --- | --- | --- | --- | --- | --- | --- |
| TPM\_ALG\_ERROR | 0x0000 |  |  |  |  | should not occur |
| TPM\_ALG\_RSA | 0x0001 | A O |  | S | IETF RFC 8017 | the RSA algorithm |
| TPM\_ALG\_TDES | 0x0003 | S |  | A | ISO/IEC 18033-3 | block cipher with various key sizes (Triple Data Encryption Algorithm, commonly called Triple Data Encryption Standard) |
| TPM\_ALG\_SHA | 0x0004 | H |  | S | ISO/IEC 10118-3 | the SHA1 algorithm |
| TPM\_ALG\_SHA1 | 0x0004 | H |  | S | ISO/IEC 10118-3 | redefinition for documentation consistency |
| TPM\_ALG\_HMAC | 0x0005 | H X |  | S | ISO/IEC 9797-2 | Hash Message Authentication Code (HMAC) algorithm |
| TPM\_ALG\_AES | 0x0006 | S |  | S | ISO/IEC 18033-3 | the AES algorithm with various key sizes |
| TPM\_ALG\_MGF1 | 0x0007 | H M |  | S | IEEE Std 1363TM-2000  IEEE Std 1363a™-2004 | hash-based mask-generation function |
| TPM\_ALG\_KEYEDHASH | 0x0008 | H O |  | S | TCG TPM 2.0 library specification | an object type that may use XOR for encryption or an HMAC for signing and may also refer to a data object that is neither signing nor encrypting |
| TPM\_ALG\_XOR | 0x000A | H S |  | S | TCG TPM 2.0 library specification | the XOR encryption algorithm |
| TPM\_ALG\_SHA256 | 0x000B | H |  | S | ISO/IEC 10118-3 | the SHA 256 algorithm |
| TPM\_ALG\_SHA384 | 0x000C | H |  | A | ISO/IEC 10118-3 | the SHA 384 algorithm |
| TPM\_ALG\_SHA512 | 0x000D | H |  | A | ISO/IEC 10118-3 | the SHA 512 algorithm |
| TPM\_ALG\_NULL | 0x0010 |  |  | S | TCG TPM 2.0 library specification | Null algorithm |
| TPM\_ALG\_SM3\_256 | 0x0012 | H |  | A | GM/T 0004-2012 | SM3 hash algorithm |
| TPM\_ALG\_SM4 | 0x0013 | S |  | A | GM/T 0002-2012 | SM4 symmetric block cipher |
| TPM\_ALG\_RSASSA | 0x0014 | A X | RSA | S | IETF RFC 8017 | a signature algorithm defined in section 8.2 (RSASSA-PKCS1-v1\_5) |
| TPM\_ALG\_RSAES | 0x0015 | A E | RSA | S | IETF RFC 8017 | a padding algorithm defined in section 7.2 (RSAES-PKCS1-v1\_5) |
| TPM\_ALG\_RSAPSS | 0x0016 | A X | RSA | S | IETF RFC 8017 | a signature algorithm defined in section 8.1 (RSASSA-PSS) |
| TPM\_ALG\_OAEP | 0x0017 | A E | RSA | S | IETF RFC 8017 | a padding algorithm defined in section 7.1 (RSAES\_OAEP) |
| TPM\_ALG\_ECDSA | 0x0018 | A X | ECC | S | ISO/IEC 14888-3 | signature algorithm using elliptic curve cryptography (ECC) |
| TPM\_ALG\_ECDH | 0x0019 | A M | ECC | S | NIST SP800-56A | secret sharing using ECC  Based on context, this can be either One-Pass Diffie-Hellman, C(1, 1, ECC CDH) defined in 6.2.2.2 or Full Unified Model C(2, 2, ECC CDH) defined in 6.1.1.2 |
| TPM\_ALG\_ECDAA | 0x001A | A X N | ECC | S | TCG TPM 2.0 library specification | elliptic-curve based, anonymous signing scheme |
| TPM\_ALG\_SM2 | 0x001B | A X | ECC | A | GM/T 0003.1–2012  GM/T 0003.2–2012  GM/T 0003.3–2012  GM/T 0003.4–2012GM/T 0003.5–2012 | SM2 – depending on context, either an elliptic-curve based, signature algorithm or a key exchange protocol  NOTE 1 Type listed as signing but, other uses are allowed according to context. |
| TPM\_ALG\_ECSCHNORR | 0x001C | A X | ECC | S | TCG TPM 2.0 library specification | elliptic-curve based Schnorr signature |
| TPM\_ALG\_ECMQV | 0x001D | A M | ECC | A | NIST SP800-56A | two-phase elliptic-curve key exchange – C(2, 2, ECC MQV) section 6.1.1.4 |
| TPM\_ALG\_KDF1\_SP800\_56A | 0x0020 | H M | ECC | S | NIST SP800-56A | concatenation key derivation function (approved alternative 1) section 5.8.1 |
| TPM\_ALG\_KDF2 | 0x0021 | H M |  | A | IEEE Std 1363a-2004 | key derivation function KDF2 section 13.2 |
| TPM\_ALG\_KDF1\_SP800\_108 | 0x0022 | H M |  | S | NIST SP800-108 | a key derivation method  Section 5.1 KDF in Counter Mode |
| TPM\_ALG\_ECC | 0x0023 | A O |  | S | ISO/IEC 15946-1 | prime field ECC |
| TPM\_ALG\_SYMCIPHER | 0x0025 | O S |  | S | TCG TPM 2.0 library specification | the object type for a symmetric block cipher |
| TPM\_ALG\_CAMELLIA | 0x0026 | S |  | A | ISO/IEC 18033-3 | Camellia is symmetric block cipher. The Camellia algorithm with various key sizes |
| TPM\_ALG\_SHA3\_256 | 0x0027 | H |  | A | NIST PUB FIPS 202 | Hash algorithm producing a 256-bit digest |
| TPM\_ALG\_SHA3\_384 | 0x0028 | H |  | A | NIST PUB FIPS 202 | Hash algorithm producing a 384-bit digest |
| TPM\_ALG\_SHA3\_512 | 0x0029 | H |  | A | NIST PUB FIPS 202 | Hash algorithm producing a 512-bit digest |
| TPM\_ALG\_CTR | 0x0040 | S E |  | A | ISO/IEC 10116 | Counter mode – if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode. |
| TPM\_ALG\_OFB | 0x0041 | S E |  | A | ISO/IEC 10116 | Output Feedback mode – if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode. |
| TPM\_ALG\_CBC | 0x0042 | S E |  | A | ISO/IEC 10116 | Cipher Block Chaining mode – if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode. |
| TPM\_ALG\_CFB | 0x0043 | S E |  | S | ISO/IEC 10116 | Cipher Feedback mode – if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode. |
| TPM\_ALG\_ECB | 0x0044 | S E |  | A | ISO/IEC 10116 | Electronic Codebook mode – if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode.  NOTE 2 This mode is not recommended for uses unless the key is frequently rotated such as in video codecs |
| reserved | 0x00C1 through 0x00C6 |  |  |  |  | 0x00C1 – 0x00C6 are reserved to prevent any overlap with the command structure tags used in TPM 1.2 |
| reserved | 0x8000 through 0xFFFF |  |  |  |  | reserved for other structure tags |

## TPM\_ECC\_CURVE

The TCG maintains a registry of all curves that have an assigned curve identifier. That registry is the definitive list of curves that may be supported by a TPM.

Table 10 is a copy of the TPM\_ECC\_CURVE constants table in the TCG registry as of the date of publication of this specification. Table 10 is provided for illustrative purposes only.

Table 10 — Definition of (UINT16) {ECC} TPM\_ECC\_CURVE Constants <IN/OUT>

| Name | Value | Comments |
| --- | --- | --- |
| +TPM\_ECC\_NONE | 0x0000 |  |
| TPM\_ECC\_NIST\_P192 | 0x0001 |  |
| TPM\_ECC\_NIST\_P224 | 0x0002 |  |
| TPM\_ECC\_NIST\_P256 | 0x0003 |  |
| TPM\_ECC\_NIST\_P384 | 0x0004 |  |
| TPM\_ECC\_NIST\_P521 | 0x0005 |  |
| TPM\_ECC\_BN\_P256 | 0x0010 | curve to support ECDAA |
| TPM\_ECC\_BN\_P638 | 0x0011 | curve to support ECDAA |
| TPM\_ECC\_SM2\_P256 | 0x0020 |  |
| TPM\_ECC\_TEST\_P192 | 0x0021 |  |
| #TPM\_RC\_CURVE |  |  |

## TPM\_CC (Command Codes)

### Format

A command is a 32-bit structure with fields assigned as shown in Figure 1. If V is SET, the command is vendor specific. If V is CLEAR, the command is not vendor specific.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3  1 | 3  0 | 2  9 | 2  8 |  |  |  |  |  |  |  |  |  |  |  | 1  6 | 1  5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0  0 |
| Res | | V | Reserved | | | | | | | | | | | | | Command Index | | | | | | | | | | | | | | | |

Figure 1 — Command Format

Table 11 — TPM Command Format Fields Description

| Bit | Name | Definition |
| --- | --- | --- |
| 15:0 | Command Index | the index of the command |
| 28:16 | Reserved | shall be zero |
| 29 | V | vendor specific |
| 31:30 | Res | shall be zero |

### TPM\_CC Listing

Table 12 lists the command codes assigned to each command name. The Dep column indicates whether the command has a dependency on the implementation of a specific algorithm.

Table 12 — Definition of (UINT32) TPM\_CC Constants (Numeric Order) <IN/OUT, S>

| Name | Command Code | Dep | Comments |
| --- | --- | --- | --- |
| TPM\_CC\_FIRST | 0x0000011F |  | Compile variable. May decrease based on implementation. |
| TPM\_CC\_NV\_UndefineSpaceSpecial | 0x0000011F |  |  |
| TPM\_CC\_EvictControl | 0x00000120 |  |  |
| TPM\_CC\_HierarchyControl | 0x00000121 |  |  |
| TPM\_CC\_NV\_UndefineSpace | 0x00000122 |  |  |
| TPM\_CC\_ChangeEPS | 0x00000124 |  |  |
| TPM\_CC\_ChangePPS | 0x00000125 |  |  |
| TPM\_CC\_Clear | 0x00000126 |  |  |
| TPM\_CC\_ClearControl | 0x00000127 |  |  |
| TPM\_CC\_ClockSet | 0x00000128 |  |  |
| TPM\_CC\_HierarchyChangeAuth | 0x00000129 |  |  |
| TPM\_CC\_NV\_DefineSpace | 0x0000012A |  |  |
| TPM\_CC\_PCR\_Allocate | 0x0000012B |  |  |
| TPM\_CC\_PCR\_SetAuthPolicy | 0x0000012C |  |  |
| TPM\_CC\_PP\_Commands | 0x0000012D |  |  |
| TPM\_CC\_SetPrimaryPolicy | 0x0000012E |  |  |
| TPM\_CC\_FieldUpgradeStart | 0x0000012F |  |  |
| TPM\_CC\_ClockRateAdjust | 0x00000130 |  |  |
| TPM\_CC\_CreatePrimary | 0x00000131 |  |  |
| TPM\_CC\_NV\_GlobalWriteLock | 0x00000132 |  |  |
| TPM\_CC\_GetCommandAuditDigest | 0x00000133 |  |  |
| TPM\_CC\_NV\_Increment | 0x00000134 |  |  |
| TPM\_CC\_NV\_SetBits | 0x00000135 |  |  |
| TPM\_CC\_NV\_Extend | 0x00000136 |  |  |
| TPM\_CC\_NV\_Write | 0x00000137 |  |  |
| TPM\_CC\_NV\_WriteLock | 0x00000138 |  |  |
| TPM\_CC\_DictionaryAttackLockReset | 0x00000139 |  |  |
| TPM\_CC\_DictionaryAttackParameters | 0x0000013A |  |  |
| TPM\_CC\_NV\_ChangeAuth | 0x0000013B |  |  |
| TPM\_CC\_PCR\_Event | 0x0000013C |  | PCR |
| TPM\_CC\_PCR\_Reset | 0x0000013D |  | PCR |
| TPM\_CC\_SequenceComplete | 0x0000013E |  |  |
| TPM\_CC\_SetAlgorithmSet | 0x0000013F |  |  |
| TPM\_CC\_SetCommandCodeAuditStatus | 0x00000140 |  |  |
| TPM\_CC\_FieldUpgradeData | 0x00000141 |  |  |
| TPM\_CC\_IncrementalSelfTest | 0x00000142 |  |  |
| TPM\_CC\_SelfTest | 0x00000143 |  |  |
| TPM\_CC\_Startup | 0x00000144 |  |  |
| TPM\_CC\_Shutdown | 0x00000145 |  |  |
| TPM\_CC\_StirRandom | 0x00000146 |  |  |
| TPM\_CC\_ActivateCredential | 0x00000147 |  |  |
| TPM\_CC\_Certify | 0x00000148 |  |  |
| TPM\_CC\_PolicyNV | 0x00000149 |  | Policy |
| TPM\_CC\_CertifyCreation | 0x0000014A |  |  |
| TPM\_CC\_Duplicate | 0x0000014B |  |  |
| TPM\_CC\_GetTime | 0x0000014C |  |  |
| TPM\_CC\_GetSessionAuditDigest | 0x0000014D |  |  |
| TPM\_CC\_NV\_Read | 0x0000014E |  |  |
| TPM\_CC\_NV\_ReadLock | 0x0000014F |  |  |
| TPM\_CC\_ObjectChangeAuth | 0x00000150 |  |  |
| TPM\_CC\_PolicySecret | 0x00000151 |  | Policy |
| TPM\_CC\_Rewrap | 0x00000152 |  |  |
| TPM\_CC\_Create | 0x00000153 |  |  |
| TPM\_CC\_ECDH\_ZGen | 0x00000154 | ECC |  |
| TPM\_CC\_HMAC | 0x00000155 | !CMAC | See NOTE 1 |
| TPM\_CC\_MAC | 0x00000155 | CMAC | See NOTE 1 |
| TPM\_CC\_Import | 0x00000156 |  |  |
| TPM\_CC\_Load | 0x00000157 |  |  |
| TPM\_CC\_Quote | 0x00000158 |  |  |
| TPM\_CC\_RSA\_Decrypt | 0x00000159 | RSA |  |
| TPM\_CC\_HMAC\_Start | 0x0000015B | !CMAC | See NOTE 1 |
| TPM\_CC\_MAC\_Start | 0x0000015B | CMAC | See NOTE 1 |
| TPM\_CC\_SequenceUpdate | 0x0000015C |  |  |
| TPM\_CC\_Sign | 0x0000015D |  |  |
| TPM\_CC\_Unseal | 0x0000015E |  |  |
| TPM\_CC\_PolicySigned | 0x00000160 |  | Policy |
| TPM\_CC\_ContextLoad | 0x00000161 |  | Context |
| TPM\_CC\_ContextSave | 0x00000162 |  | Context |
| TPM\_CC\_ECDH\_KeyGen | 0x00000163 | ECC |  |
| TPM\_CC\_EncryptDecrypt | 0x00000164 |  |  |
| TPM\_CC\_FlushContext | 0x00000165 |  | Context |
| TPM\_CC\_LoadExternal | 0x00000167 |  |  |
| TPM\_CC\_MakeCredential | 0x00000168 |  |  |
| TPM\_CC\_NV\_ReadPublic | 0x00000169 |  | NV |
| TPM\_CC\_PolicyAuthorize | 0x0000016A |  | Policy |
| TPM\_CC\_PolicyAuthValue | 0x0000016B |  | Policy |
| TPM\_CC\_PolicyCommandCode | 0x0000016C |  | Policy |
| TPM\_CC\_PolicyCounterTimer | 0x0000016D |  | Policy |
| TPM\_CC\_PolicyCpHash | 0x0000016E |  | Policy |
| TPM\_CC\_PolicyLocality | 0x0000016F |  | Policy |
| TPM\_CC\_PolicyNameHash | 0x00000170 |  | Policy |
| TPM\_CC\_PolicyOR | 0x00000171 |  | Policy |
| TPM\_CC\_PolicyTicket | 0x00000172 |  | Policy |
| TPM\_CC\_ReadPublic | 0x00000173 |  |  |
| TPM\_CC\_RSA\_Encrypt | 0x00000174 | RSA |  |
| TPM\_CC\_StartAuthSession | 0x00000176 |  |  |
| TPM\_CC\_VerifySignature | 0x00000177 |  |  |
| TPM\_CC\_ECC\_Parameters | 0x00000178 | ECC |  |
| TPM\_CC\_FirmwareRead | 0x00000179 |  |  |
| TPM\_CC\_GetCapability | 0x0000017A |  |  |
| TPM\_CC\_GetRandom | 0x0000017B |  |  |
| TPM\_CC\_GetTestResult | 0x0000017C |  |  |
| TPM\_CC\_Hash | 0x0000017D |  |  |
| TPM\_CC\_PCR\_Read | 0x0000017E |  | PCR |
| TPM\_CC\_PolicyPCR | 0x0000017F |  | Policy |
| TPM\_CC\_PolicyRestart | 0x00000180 |  |  |
| TPM\_CC\_ReadClock | 0x00000181 |  |  |
| TPM\_CC\_PCR\_Extend | 0x00000182 |  |  |
| TPM\_CC\_PCR\_SetAuthValue | 0x00000183 |  |  |
| TPM\_CC\_NV\_Certify | 0x00000184 |  |  |
| TPM\_CC\_EventSequenceComplete | 0x00000185 |  |  |
| TPM\_CC\_HashSequenceStart | 0x00000186 |  |  |
| TPM\_CC\_PolicyPhysicalPresence | 0x00000187 |  | Policy |
| TPM\_CC\_PolicyDuplicationSelect | 0x00000188 |  | Policy |
| TPM\_CC\_PolicyGetDigest | 0x00000189 |  | Policy |
| TPM\_CC\_TestParms | 0x0000018A |  |  |
| TPM\_CC\_Commit | 0x0000018B | ECC |  |
| TPM\_CC\_PolicyPassword | 0x0000018C |  | Policy |
| TPM\_CC\_ZGen\_2Phase | 0x0000018D | ECC |  |
| TPM\_CC\_EC\_Ephemeral | 0x0000018E | ECC |  |
| TPM\_CC\_PolicyNvWritten | 0x0000018F |  | Policy |
| TPM\_CC\_PolicyTemplate | 0x00000190 |  | Policy |
| TPM\_CC\_CreateLoaded | 0x00000191 |  |  |
| TPM\_CC\_PolicyAuthorizeNV | 0x00000192 |  | Policy |
| TPM\_CC\_EncryptDecrypt2 | 0x00000193 |  |  |
| TPM\_CC\_AC\_GetCapability | 0x00000194 |  |  |
| TPM\_CC\_AC\_Send | 0x00000195 |  |  |
| TPM\_CC\_Policy\_AC\_SendSelect | 0x00000196 |  | Policy |
| TPM\_CC\_CertifyX509 | 0x00000197 |  |  |
| TPM\_CC\_ACT\_SetTimeout | 0x00000198 |  |  |
| TPM\_CC\_ECC\_Encrypt | 0x00000199 | ECC |  |
| TPM\_CC\_ECC\_Decrypt | 0x0000019A | ECC |  |
| TPM\_CC\_LAST | 0x0000019A |  | Compile variable. May increase based on implementation. |
| CC\_VEND | 0x20000000 |  |  |
| TPM\_CC\_Vendor\_TCG\_Test | CC\_VEND+0x0000 |  | Used for testing of command dispatch |
| #TPM\_RC\_COMMAND\_CODE |  |  |  |

NOTE 1 A TPM may implement either TPM2\_HMAC()/TPM2\_HMAC\_Start() or TPM2\_MAC()/TPM2\_MAC\_Start() but not both, as they have the same command code and there is no way to distinguish them. A TPM that supports TPM2\_MAC()/TPM2\_MAC\_Start() will support any code that was written to use TPM2\_HMAC()/TPM2\_HMAC\_Start(), but a TPM that supports TPM2\_HMAC()/TPM2\_HMAC\_Start() will not support a MAC based on symmetric block ciphers.

## TPM\_RC (Response Codes)

### Description

Each return from the TPM has a 32-bit response code. The TPM will always set the upper 20 bits (31:12) of the response code to 0 00 0016 and the low-order 12 bits (11:00) will contain the response code.

When a command succeeds, the TPM shall return TPM\_RC\_SUCCESS (0 0016) and will update any authorization-session nonce associated with the command.

When a command fails to complete for any reason, the TPM shall return

* a TPM\_ST (UINT16) with a value of TPM\_TAG\_RSP\_COMMAND or TPM\_ST\_NO\_SESSIONS, followed by
* a UINT32 (*responseSize*) with a value of 10, followed by
* a UINT32 containing a response code with a value other than TPM\_RC\_SUCCESS.

Commands defined in this specification will use a tag of either TPM\_ST\_NO\_SESSIONS or TPM\_ST\_SESSIONS. Error responses will use a tag value of TPM\_ST\_NO\_SESSIONS and the response code will be as defined in this specification. Commands that use tags defined in the TPM 1.2 specification will use TPM\_TAG\_RSP\_COMMAND in an error and a response code defined in TPM 1.2.

If the tag of the command is not a recognized command tag, the TPM error response will differ depending on TPM 1.2 compatibility. If the TPM supports 1.2 compatibility, the TPM shall return a tag of TPM\_TAG\_RSP\_COMMAND and an appropriate TPM 1.2 response code (TPM\_BADTAG = 00 00 00 1E16). If the TPM does not have compatibility with TPM 1.2, the TPM shall return TPM\_ST\_NO\_SESSION and a response code of TPM\_RC\_TAG.

When a command fails, the TPM shall not update the authorization-session nonces associated with the command and will not close the authorization sessions used by the command. Audit digests will not be updated on an error. Unless noted in the command actions, a command that returns an error shall leave the state of the TPM as if the command had not been attempted. The exception to this principle is that a failure due to an authorization failure may update the dictionary-attack protection values.

### Response Code Formats

The response codes for this specification are defined such that there is no overlap between the response codes used for this specification and those assigned in previous TPM specifications.

The formats defined in this clause only apply when the tag for the response is TPM\_ST\_NO\_SESSIONS.

The response codes use two different format groups. One group contains the TPM 1.2 compatible response codes and the response codes for this specification that are not related to command parameters. The second group contains the errors that may be associated with a command parameter, handle, or session.

Figure 2 shows the format for the response codes when bit 7 is zero.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 1  1 | 1  0 | 0  9 | 0  8 | 0  7 | 0  6 | 0  5 | 0  4 | 0  3 | 0  2 | 0  1 | 0  0 |
|  | S | T | r | V | F | E | | | | | | |

Figure 2 — Format-Zero Response Codes

The field definitions are:

Table 13 — Format-Zero Response Codes

| Bit | Name | Definition |
| --- | --- | --- |
| 06:00 | E | the error number  The interpretation of this field is dependent on the setting of the F and S fields. |
| 07 | F | format selector  **CLEAR** when the format is as defined in this Table 13 or when the response code is TPM\_RC\_BAD\_TAG. |
| 08 | V | version  **SET (1)**: The error number is defined in this specification and is returned when the response tag is TPM\_ST\_NO\_SESSIONS.  **CLEAR (0)**: The error number is defined by a previous TPM specification. The error number is returned when the response tag is TPM\_TAG\_RSP\_COMMAND.  NOTE In any error number returned by a TPM, the F (bit 7) and V (bit 8) attributes shall be CLEAR when the response tag is TPM\_TAG\_RSP\_COMMAND value used in TPM 1.2. |
| 09 | Reserved | shall be zero. |
| 10 | T | TCG/Vendor indicator  **SET (1):** The response code is defined by the TPM vendor.  **CLEAR (0):** The response code is defined by the TCG (a value in this specification).  NOTE This attribute does not indicate a vendor-specific code unless the *F* attribute (bit[07]) is CLEAR. |
| 11 | S | severity  **SET (1)**: The response code is a warning and the command was not necessarily in error. This command indicates that the TPM is busy or that the resources of the TPM have to be adjusted in order to allow the command to execute.  **CLEAR (0)**: The response code indicates that the command had an error that would prevent it from running. |

When the format bit (bit 7) is SET, then the error occurred during the unmarshaling or validation of an input parameter to the TPM. Figure 3 shows the format for the response codes when bit 7 is one.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| bit | 1  1 | 1  0 | 0  9 | 0  8 | 0  7 | 0  6 | 0  5 | 0  4 | 0  3 | 0  2 | 0  1 | 0  0 |
|  | N | | | | 1 | P | E | | | | | |

Figure 3 — Format-One Response Codes

There are 64 errors with this format. The errors can be associated with a parameter, handle, or session. The error number for this format is in bits[05:00]. When an error is associated with a parameter, TPM\_RC\_P (0 4016) is added and N is set to the parameter number.

NOTE 1 In the reference implementation, for a RC\_FMT1 response code, a constant of the form RC\_Command\_parameterName is the one based parameter number (TPM\_RC\_n) plus TPM\_RC\_P.

Example RC\_Startup\_startupType is the first parameter, TPM\_RC\_1 (0x100) plus TPM\_RC\_P (0x40) or 0x140. TPM\_RC\_VALUE (RC\_FMT1 (0x080) + 0x004) + RC\_Startup\_startupType is thus 0x080 + 0x004 + 0x140 = 0x1c4.

For an error associated with a handle, a parameter number (1 to 7) is added to the N field. For an error associated with a session, a value of 8 plus the session number (1 to 7) is added to the N field. In other words, if P is clear, then a value of 0 to 7 in the N field will indicate a handle error, and a value of 8 – 15 will indicate a session error.

NOTE 2 If an implementation is not able to designate the handle, session, or parameter in error, then P and N will be zero.

The field definitions are:

Table 14 — Format-One Response Codes

| Bit | Name | Definition |
| --- | --- | --- |
| 05:00 | E | the error number  The error number is independent of the other settings. |
| 06 | P | **SET (1):** The error is associated with a parameter.  **CLEAR (0):** The error is associated with a handle or a session. |
| 07 | F | the response code format selector  This field shall be SET for the format in this table. |
| 11:08 | N | the number of the handle, session, or parameter in error. The number is one based. See TPM\_RC\_1 through TPM\_RC\_F.  If P is SET, then this field is the parameter in error.  If P is CLEAR, then this field indicates the handle or session in error. Handles use values of N between 00002 and 01112. Sessions use values between 10002 and 11112.  NOTE Bit 11 distinguishes between handles and sessions. Bits 10:8 0002 indicate that the number is unspecified. |

The groupings of response codes are determined by bits 08, 07, and 06 of the response code as summarized in Table 15.

Table 15 — Response Code Groupings

| Bit | | | Definition |
| --- | --- | --- | --- |
| 08 | 07 | 06 |
| 0 | 0 | x | a response code defined by TPM 1.2  NOTE An “x” in a column indicates that this may be either 0 or 1 and not affect the grouping of the response code. |
| 1 | 0 | x | a response code defined by this specification with no handle, session, or parameter number modifier |
| x | 1 | 0 | a response code defined by this specification with either a handle or session number modifier |
| x | 1 | 1 | a response code defined by this specification with a parameter number modifier |

### TPM\_RC Values

In general, response codes defined in TPM 2.0 Part 2 will be unmarshaling errors and will have the F (format) bit SET. Codes that are unique to TPM 2.0 Part 3 will have the F bit CLEAR but the V (version) attribute will be SET to indicate that it is a TPM 2.0 response code. See *Response Code Details* in TPM 2.0 Part 1.

NOTE The constant RC\_VER1 is used to indicate that the V attribute is SET and the constant RC\_FMT1 is used to indicate that the F attribute is SET and that the return code is variable based on handle, session, and parameter modifiers.

Table 16 — Definition of (UINT32) TPM\_RC Constants (Actions) <OUT>

| Name | Value | Description |
| --- | --- | --- |
| TPM\_RC\_SUCCESS | 0x000 |  |
| TPM\_RC\_BAD\_TAG | 0x01E | defined for compatibility with TPM 1.2 |
| RC\_VER1 | 0x100 | set for all format 0 response codes |
| TPM\_RC\_INITIALIZE | RC\_VER1 + 0x000 | TPM not initialized by TPM2\_Startup or already initialized |
| TPM\_RC\_FAILURE | RC\_VER1 + 0x001 | commands not being accepted because of a TPM failure  NOTE This may be returned by TPM2\_GetTestResult() as the *testResult* parameter. |
| TPM\_RC\_SEQUENCE | RC\_VER1 + 0x003 | improper use of a sequence handle |
| TPM\_RC\_PRIVATE | RC\_VER1 + 0x00B | not currently used |
| TPM\_RC\_HMAC | RC\_VER1 + 0x019 | not currently used |
| TPM\_RC\_DISABLED | RC\_VER1 + 0x020 | the command is disabled |
| TPM\_RC\_EXCLUSIVE | RC\_VER1 + 0x021 | command failed because audit sequence required exclusivity |
| TPM\_RC\_AUTH\_TYPE | RC\_VER1 + 0x024 | authorization handle is not correct for command |
| TPM\_RC\_AUTH\_MISSING | RC\_VER1 + 0x025 | command requires an authorization session for handle and it is not present. |
| TPM\_RC\_POLICY | RC\_VER1 + 0x026 | policy failure in math operation or an invalid authPolicy value |
| TPM\_RC\_PCR | RC\_VER1 + 0x027 | PCR check fail |
| TPM\_RC\_PCR\_CHANGED | RC\_VER1 + 0x028 | PCR have changed since checked. |
| TPM\_RC\_UPGRADE | RC\_VER1 + 0x02D | for all commands other than TPM2\_FieldUpgradeData(), this code indicates that the TPM is in field upgrade mode; for TPM2\_FieldUpgradeData(), this code indicates that the TPM is not in field upgrade mode |
| TPM\_RC\_TOO\_MANY\_CONTEXTS | RC\_VER1 + 0x02E | context ID counter is at maximum. |
| TPM\_RC\_AUTH\_UNAVAILABLE | RC\_VER1 + 0x02F | authValue or authPolicy is not available for selected entity. |
| TPM\_RC\_REBOOT | RC\_VER1 + 0x030 | a \_TPM\_Init and Startup(CLEAR) is required before the TPM can resume operation. |
| TPM\_RC\_UNBALANCED | RC\_VER1 + 0x031 | the protection algorithms (hash and symmetric) are not reasonably balanced. The digest size of the hash must be larger than the key size of the symmetric algorithm. |
| TPM\_RC\_COMMAND\_SIZE | RC\_VER1 + 0x042 | command *commandSize* value is inconsistent with contents of the command buffer; either the size is not the same as the octets loaded by the hardware interface layer or the value is not large enough to hold a command header |
| TPM\_RC\_COMMAND\_CODE | RC\_VER1 + 0x043 | command code not supported |
| TPM\_RC\_AUTHSIZE | RC\_VER1 + 0x044 | the value of *authorizationSize* is out of range or the number of octets in the Authorization Area is greater than required |
| TPM\_RC\_AUTH\_CONTEXT | RC\_VER1 + 0x045 | use of an authorization session with a context command or another command that cannot have an authorization session. |
| TPM\_RC\_NV\_RANGE | RC\_VER1 + 0x046 | NV offset+size is out of range. |
| TPM\_RC\_NV\_SIZE | RC\_VER1 + 0x047 | Requested allocation size is larger than allowed. |
| TPM\_RC\_NV\_LOCKED | RC\_VER1 + 0x048 | NV access locked. |
| TPM\_RC\_NV\_AUTHORIZATION | RC\_VER1 + 0x049 | NV access authorization fails in command actions (this failure does not affect lockout.action) |
| TPM\_RC\_NV\_UNINITIALIZED | RC\_VER1 + 0x04A | an NV Index is used before being initialized or the state saved by TPM2\_Shutdown(STATE) could not be restored |
| TPM\_RC\_NV\_SPACE | RC\_VER1 + 0x04B | insufficient space for NV allocation |
| TPM\_RC\_NV\_DEFINED | RC\_VER1 + 0x04C | NV Index or persistent object already defined |
| TPM\_RC\_BAD\_CONTEXT | RC\_VER1 + 0x050 | context in TPM2\_ContextLoad() is not valid |
| TPM\_RC\_CPHASH | RC\_VER1 + 0x051 | cpHash value already set or not correct for use |
| TPM\_RC\_PARENT | RC\_VER1 + 0x052 | handle for parent is not a valid parent |
| TPM\_RC\_NEEDS\_TEST | RC\_VER1 + 0x053 | some function needs testing. |
| TPM\_RC\_NO\_RESULT | RC\_VER1 + 0x054 | returned when an internal function cannot process a request due to an unspecified problem. This code is usually related to invalid parameters that are not properly filtered by the input unmarshaling code. |
| TPM\_RC\_SENSITIVE | RC\_VER1 + 0x055 | the sensitive area did not unmarshal correctly after decryption – this code is used in lieu of the other unmarshaling errors so that an attacker cannot determine where the unmarshaling error occurred |
| RC\_MAX\_FM0 | RC\_VER1 + 0x07F | largest version 1 code that is not a warning |
|  |  | New Subsection |
| RC\_FMT1 | 0x080 | This bit is SET in all format 1 response codes  The codes in this group may have a value added to them to indicate the handle, session, or parameter to which they apply. |
| TPM\_RC\_ASYMMETRIC | RC\_FMT1 + 0x001 | asymmetric algorithm not supported or not correct |
| TPM\_RC\_ATTRIBUTES | RC\_FMT1 + 0x002 | inconsistent attributes |
| TPM\_RC\_HASH | RC\_FMT1 + 0x003 | hash algorithm not supported or not appropriate |
| TPM\_RC\_VALUE | RC\_FMT1 + 0x004 | value is out of range or is not correct for the context |
| TPM\_RC\_HIERARCHY | RC\_FMT1 + 0x005 | hierarchy is not enabled or is not correct for the use |
| TPM\_RC\_KEY\_SIZE | RC\_FMT1 + 0x007 | key size is not supported |
| TPM\_RC\_MGF | RC\_FMT1 + 0x008 | mask generation function not supported |
| TPM\_RC\_MODE | RC\_FMT1 + 0x009 | mode of operation not supported |
| TPM\_RC\_TYPE | RC\_FMT1 + 0x00A | the type of the value is not appropriate for the use |
| TPM\_RC\_HANDLE | RC\_FMT1 + 0x00B | the handle is not correct for the use |
| TPM\_RC\_KDF | RC\_FMT1 + 0x00C | unsupported key derivation function or function not appropriate for use |
| TPM\_RC\_RANGE | RC\_FMT1 + 0x00D | value was out of allowed range. |
| TPM\_RC\_AUTH\_FAIL | RC\_FMT1 + 0x00E | the authorization HMAC check failed and DA counter incremented |
| TPM\_RC\_NONCE | RC\_FMT1 + 0x00F | invalid nonce size or nonce value mismatch |
| TPM\_RC\_PP | RC\_FMT1 + 0x010 | authorization requires assertion of PP |
| TPM\_RC\_SCHEME | RC\_FMT1 + 0x012 | unsupported or incompatible scheme |
| TPM\_RC\_SIZE | RC\_FMT1 + 0x015 | structure is the wrong size |
| TPM\_RC\_SYMMETRIC | RC\_FMT1 + 0x016 | unsupported symmetric algorithm or key size, or not appropriate for instance |
| TPM\_RC\_TAG | RC\_FMT1 + 0x017 | incorrect structure tag |
| TPM\_RC\_SELECTOR | RC\_FMT1 + 0x018 | union selector is incorrect |
| TPM\_RC\_INSUFFICIENT | RC\_FMT1 + 0x01A | the TPM was unable to unmarshal a value because there were not enough octets in the input buffer |
| TPM\_RC\_SIGNATURE | RC\_FMT1 + 0x01B | the signature is not valid |
| TPM\_RC\_KEY | RC\_FMT1 + 0x01C | key fields are not compatible with the selected use |
| TPM\_RC\_POLICY\_FAIL | RC\_FMT1 + 0x01D | a policy check failed |
| TPM\_RC\_INTEGRITY | RC\_FMT1 + 0x01F | integrity check failed |
| TPM\_RC\_TICKET | RC\_FMT1 + 0x020 | invalid ticket |
| TPM\_RC\_RESERVED\_BITS | RC\_FMT1 + 0x021 | reserved bits not set to zero as required |
| TPM\_RC\_BAD\_AUTH | RC\_FMT1 + 0x022 | authorization failure without DA implications |
| TPM\_RC\_EXPIRED | RC\_FMT1 + 0x023 | the policy has expired |
| TPM\_RC\_POLICY\_CC | RC\_FMT1 + 0x024 | the *commandCode* in the policy is not the *commandCode* of the command or the command code in a policy command references a command that is not implemented |
| TPM\_RC\_BINDING | RC\_FMT1 + 0x025 | public and sensitive portions of an object are not cryptographically bound |
| TPM\_RC\_CURVE | RC\_FMT1 + 0x026 | curve not supported |
| TPM\_RC\_ECC\_POINT | RC\_FMT1 + 0x027 | point is not on the required curve. |
|  |  |  |
|  |  | New Subsection |
| RC\_WARN | 0x900 | set for warning response codes |
| TPM\_RC\_CONTEXT\_GAP | RC\_WARN + 0x001 | gap for context ID is too large |
| TPM\_RC\_OBJECT\_MEMORY | RC\_WARN + 0x002 | out of memory for object contexts |
| TPM\_RC\_SESSION\_MEMORY | RC\_WARN + 0x003 | out of memory for session contexts |
| TPM\_RC\_MEMORY | RC\_WARN + 0x004 | out of shared object/session memory or need space for internal operations |
| TPM\_RC\_SESSION\_HANDLES | RC\_WARN + 0x005 | out of session handles – a session must be flushed before a new session may be created |
| TPM\_RC\_OBJECT\_HANDLES | RC\_WARN + 0x006 | out of object handles – the handle space for objects is depleted and a reboot is required  NOTE 1 This cannot occur on the reference implementation.  NOTE 2 There is no reason why an implementation would implement a design that would deplete handle space. Platform specifications are encouraged to forbid it. |
| TPM\_RC\_LOCALITY | RC\_WARN + 0x007 | bad locality |
| TPM\_RC\_YIELDED | RC\_WARN + 0x008 | the TPM has suspended operation on the command; forward progress was made and the command may be retried  See TPM 2.0 Part 1, “Multi-tasking.”  NOTE This cannot occur on the reference implementation. |
| TPM\_RC\_CANCELED | RC\_WARN + 0x009 | the command was canceled |
| TPM\_RC\_TESTING | RC\_WARN + 0x00A | TPM is performing self-tests |
| TPM\_RC\_REFERENCE\_H0 | RC\_WARN + 0x010 | the 1st handle in the handle area references a transient object or session that is not loaded |
| TPM\_RC\_REFERENCE\_H1 | RC\_WARN + 0x011 | the 2nd handle in the handle area references a transient object or session that is not loaded |
| TPM\_RC\_REFERENCE\_H2 | RC\_WARN + 0x012 | the 3rd handle in the handle area references a transient object or session that is not loaded |
| TPM\_RC\_REFERENCE\_H3 | RC\_WARN + 0x013 | the 4th handle in the handle area references a transient object or session that is not loaded |
| TPM\_RC\_REFERENCE\_H4 | RC\_WARN + 0x014 | the 5th handle in the handle area references a transient object or session that is not loaded |
| TPM\_RC\_REFERENCE\_H5 | RC\_WARN + 0x015 | the 6th handle in the handle area references a transient object or session that is not loaded |
| TPM\_RC\_REFERENCE\_H6 | RC\_WARN + 0x016 | the 7th handle in the handle area references a transient object or session that is not loaded |
| TPM\_RC\_REFERENCE\_S0 | RC\_WARN + 0x018 | the 1st authorization session handle references a session that is not loaded |
| TPM\_RC\_REFERENCE\_S1 | RC\_WARN + 0x019 | the 2nd authorization session handle references a session that is not loaded |
| TPM\_RC\_REFERENCE\_S2 | RC\_WARN + 0x01A | the 3rd authorization session handle references a session that is not loaded |
| TPM\_RC\_REFERENCE\_S3 | RC\_WARN + 0x01B | the 4th authorization session handle references a session that is not loaded |
| TPM\_RC\_REFERENCE\_S4 | RC\_WARN + 0x01C | the 5th session handle references a session that is not loaded |
| TPM\_RC\_REFERENCE\_S5 | RC\_WARN + 0x01D | the 6th session handle references a session that is not loaded |
| TPM\_RC\_REFERENCE\_S6 | RC\_WARN + 0x01E | the 7th authorization session handle references a session that is not loaded |
| TPM\_RC\_NV\_RATE | RC\_WARN + 0x020 | the TPM is rate-limiting accesses to prevent wearout of NV |
| TPM\_RC\_LOCKOUT | RC\_WARN + 0x021 | authorizations for objects subject to DA protection are not allowed at this time because the TPM is in DA lockout mode |
| TPM\_RC\_RETRY | RC\_WARN + 0x022 | the TPM was not able to start the command |
| TPM\_RC\_NV\_UNAVAILABLE | RC\_WARN + 0x023 | the command may require writing of NV and NV is not current accessible |
| TPM\_RC\_NOT\_USED | RC\_WARN + 0x7F | this value is reserved and shall not be returned by the TPM |
|  |  | Additional Defines |
| TPM\_RC\_H | 0x000 | add to a handle-related error |
| TPM\_RC\_P | 0x040 | add to a parameter-related error |
| TPM\_RC\_S | 0x800 | add to a session-related error |
| TPM\_RC\_1 | 0x100 | add to a parameter-, handle-, or session-related error |
| TPM\_RC\_2 | 0x200 | add to a parameter-, handle-, or session-related error |
| TPM\_RC\_3 | 0x300 | add to a parameter-, handle-, or session-related error |
| TPM\_RC\_4 | 0x400 | add to a parameter-, handle-, or session-related error |
| TPM\_RC\_5 | 0x500 | add to a parameter-, handle-, or session-related error |
| TPM\_RC\_6 | 0x600 | add to a parameter-, handle-, or session-related error |
| TPM\_RC\_7 | 0x700 | add to a parameter-, handle-, or session-related error |
| TPM\_RC\_8 | 0x800 | add to a parameter-related error |
| TPM\_RC\_9 | 0x900 | add to a parameter-related error |
| TPM\_RC\_A | 0xA00 | add to a parameter-related error |
| TPM\_RC\_B | 0xB00 | add to a parameter-related error |
| TPM\_RC\_C | 0xC00 | add to a parameter-related error |
| TPM\_RC\_D | 0xD00 | add to a parameter-related error |
| TPM\_RC\_E | 0xE00 | add to a parameter-related error |
| TPM\_RC\_F | 0xF00 | add to a parameter-related error |
| TPM\_RC\_N\_MASK | 0xF00 | number mask |

## TPM\_CLOCK\_ADJUST

A TPM\_CLOCK\_ADJUST value is used to change the rate at which the TPM internal oscillator is divided. A change to the divider will change the rate at which *Clock* and *Time* change.

NOTE The recommended adjustments are approximately 1% for a course adjustment, 0.1% for a medium adjustment, and the minimum possible on the implementation for the fine adjustment (e.g., one count of the pre-scalar if possible).

Table 17 — Definition of (INT8) TPM\_CLOCK\_ADJUST Constants <IN>

|  |  |  |
| --- | --- | --- |
| Name | Value | Comments |
| TPM\_CLOCK\_COARSE\_SLOWER | -3 | Slow the *Clock* update rate by one coarse adjustment step. |
| TPM\_CLOCK\_MEDIUM\_SLOWER | -2 | Slow the *Clock* update rate by one medium adjustment step. |
| TPM\_CLOCK\_FINE\_SLOWER | -1 | Slow the *Clock* update rate by one fine adjustment step. |
| TPM\_CLOCK\_NO\_CHANGE | 0 | No change to the *Clock* update rate. |
| TPM\_CLOCK\_FINE\_FASTER | 1 | Speed the *Clock* update rate by one fine adjustment step. |
| TPM\_CLOCK\_MEDIUM\_FASTER | 2 | Speed the *Clock* update rate by one medium adjustment step. |
| TPM\_CLOCK\_COARSE\_FASTER | 3 | Speed the *Clock* update rate by one coarse adjustment step. |
| #TPM\_RC\_VALUE |  |  |

## TPM\_EO (EA Arithmetic Operands)

Table 18 — Definition of (UINT16) TPM\_EO Constants <IN/OUT>

|  |  |  |
| --- | --- | --- |
| Operation Name | Value | Comments |
| TPM\_EO\_EQ | 0x0000 | A = B |
| TPM\_EO\_NEQ | 0x0001 | A ≠ B |
| TPM\_EO\_SIGNED\_GT | 0x0002 | A > B signed |
| TPM\_EO\_UNSIGNED\_GT | 0x0003 | A > B unsigned |
| TPM\_EO\_SIGNED\_LT | 0x0004 | A < B signed |
| TPM\_EO\_UNSIGNED\_LT | 0x0005 | A < B unsigned |
| TPM\_EO\_SIGNED\_GE | 0x0006 | A ≥ B signed |
| TPM\_EO\_UNSIGNED\_GE | 0x0007 | A ≥ B unsigned |
| TPM\_EO\_SIGNED\_LE | 0x0008 | A ≤ B signed |
| TPM\_EO\_UNSIGNED\_LE | 0x0009 | A ≤ B unsigned |
| TPM\_EO\_BITSET | 0x000A | All bits SET in B are SET in A. ((A&B)=B) |
| TPM\_EO\_BITCLEAR | 0x000B | All bits SET in B are CLEAR in A. ((A&B)=0) |
| #TPM\_RC\_VALUE |  | Response code returned when unmarshaling of this type fails |

## TPM\_ST (Structure Tags)

Structure tags are used to disambiguate structures. They are 16-bit values with the most significant bit SET so that they do not overlap TPM\_ALG\_ID values. A single exception is made for the value associated with TPM\_ST\_RSP\_COMMAND (0x00C4), which has the same value as the TPM\_TAG\_RSP\_COMMAND tag from earlier versions of this specification. This value is used when the TPM is compatible with a previous TPM specification and the TPM cannot determine which family of response code to return because the command tag is not valid.

Many of the structures defined in this document have parameters that are unions of other structures. That is, a parameter may be one of several structures. The parameter will have a selector value that indicates which of the options is actually present.

In order to allow the marshaling and unmarshaling code to determine which of the possible structures is allowed, each selector will have a unique interface type and will constrain the number of possible tag values.

Table 19 defines the structure tags values. The definition of many structures is context-sensitive using an algorithm ID. In cases where an algorithm ID is not a meaningful way to designate the structure, the values in this table are used.

Table 19 — Definition of (UINT16) TPM\_ST Constants <IN/OUT, S>

| Name | Value | Comments |
| --- | --- | --- |
| TPM\_ST\_RSP\_COMMAND | 0x00C4 | *tag* value for a response; used when there is an error in the tag. This is also the value returned from a TPM 1.2 when an error occurs. This value is used in this specification because an error in the command tag may prevent determination of the family. When this tag is used in the response, the response code will be TPM\_RC\_BAD\_TAG (0 1E16), which has the same numeric value as the TPM 1.2 response code for TPM\_BADTAG.  NOTE In a previously published version of this specification, TPM\_RC\_BAD\_TAG was incorrectly assigned a value of 0x030 instead of 30 (0x01e). Some implementations my return the old value instead of the new value. |
| TPM\_ST\_NULL | 0X8000 | no structure type specified |
| TPM\_ST\_NO\_SESSIONS | 0x8001 | *tag* value for a command/response for a command defined in this specification; indicating that the command/response has no attached sessions and no *authorizationSize*/*parameterSize* value is present  If the *responseCode* from the TPM is not TPM\_RC\_SUCCESS, then the response tag shall have this value. |
| TPM\_ST\_SESSIONS | 0x8002 | *tag* value for a command/response for a command defined in this specification; indicating that the command/response has one or more attached sessions and the *authorizationSize*/*parameterSize* field is present |
| reserved | 0x8003 | When used between application software and the TPM resource manager, this tag indicates that the command has no sessions and the handles are using the Name format rather than the 32-bit handle format.  NOTE 1 The response to application software will have a *tag* of TPM\_ST\_NO\_SESSIONS.  Between the TRM and TPM, this tag would occur in a response from a TPM that overlaps the *tag* parameter of a request with the *tag* parameter of a response, when the response has no associated sessions.  NOTE 2 This tag is not used by all TPM or TRM implementations. |
| reserved | 0x8004 | When used between application software and the TPM resource manager, this tag indicates that the command has sessions and the handles are using the Name format rather than the 32-bit handle format.  NOTE 1 If the command completes successfully, the response to application software will have a *tag* of TPM\_ST\_SESSIONS.  Between the TRM and TPM, would occur in a response from a TPM that overlaps the *tag* parameter of a request with the *tag* parameter of a response, when the response has authorization sessions.  NOTE 2 This tag is not used by all TPM or TRM implementations. |
| TPM\_ST\_ATTEST\_NV | 0x8014 | tag for an attestation structure |
| TPM\_ST\_ATTEST\_COMMAND\_AUDIT | 0x8015 | tag for an attestation structure |
| TPM\_ST\_ATTEST\_SESSION\_AUDIT | 0x8016 | tag for an attestation structure |
| TPM\_ST\_ATTEST\_CERTIFY | 0x8017 | tag for an attestation structure |
| TPM\_ST\_ATTEST\_QUOTE | 0x8018 | tag for an attestation structure |
| TPM\_ST\_ATTEST\_TIME | 0x8019 | tag for an attestation structure |
| TPM\_ST\_ATTEST\_CREATION | 0x801A | tag for an attestation structure |
| reserved | 0x801B | do not use  NOTE This was previously assigned to TPM\_ST\_ATTEST\_NV. The tag is changed because the structure has changed |
| TPM\_ST\_ATTEST\_NV\_DIGEST | 0x801C | tag for an attestation structure |
| TPM\_ST\_CREATION | 0x8021 | tag for a ticket type |
| TPM\_ST\_VERIFIED | 0x8022 | tag for a ticket type |
| TPM\_ST\_AUTH\_SECRET | 0x8023 | tag for a ticket type |
| TPM\_ST\_HASHCHECK | 0x8024 | tag for a ticket type |
| TPM\_ST\_AUTH\_SIGNED | 0x8025 | tag for a ticket type |
| TPM\_ST\_FU\_MANIFEST | 0x8029 | tag for a structure describing a Field Upgrade Policy |

## TPM\_SU (Startup Type)

These values are used in TPM2\_Startup() to indicate the shutdown and startup mode. The defined startup sequences are:

1. TPM Reset – Two cases:
   1. Shutdown(CLEAR) followed by Startup(CLEAR)
   2. Startup(CLEAR) with no Shutdown()
2. TPM Restart – Shutdown(STATE) followed by Startup(CLEAR)
3. TPM Resume – Shutdown(STATE) followed by Startup(STATE)

TPM\_SU values of 80 0016 and above are reserved for internal use of the TPM and may not be assigned values.

NOTE In the reference code, a value of FF FF16 indicates that the startup state has not been set. If this was defined in this table to be, say, TPM\_SU\_NONE, then TPM\_SU\_NONE would be a valid input value but the caller is not allowed to indicate the that the startup type is TPM\_SU\_NONE so the reserved value is defined in the implementation as required for internal TPM uses.

Table 20 — Definition of (UINT16) TPM\_SU Constants <IN>

|  |  |  |
| --- | --- | --- |
| Name | Value | Description |
| TPM\_SU\_CLEAR | 0x0000 | on TPM2\_Shutdown(), indicates that the TPM should prepare for loss of power and save state required for an orderly startup (TPM Reset).  on TPM2\_Startup(), indicates that the TPM should perform TPM Reset or TPM Restart |
| TPM\_SU\_STATE | 0x0001 | on TPM2\_Shutdown(), indicates that the TPM should prepare for loss of power and save state required for an orderly startup (TPM Restart or TPM Resume)  on TPM2\_Startup(), indicates that the TPM should restore the state saved by TPM2\_Shutdown(TPM\_SU\_STATE) |
| #TPM\_RC\_VALUE |  | response code when incorrect value is used |

## TPM\_SE (Session Type)

This type is used in TPM2\_StartAuthSession() to indicate the type of the session to be created.

Table 21 — Definition of (UINT8) TPM\_SE Constants <IN>

| Name | Value | Description |
| --- | --- | --- |
| TPM\_SE\_HMAC | 0x00 |  |
| TPM\_SE\_POLICY | 0x01 |  |
| TPM\_SE\_TRIAL | 0x03 | The policy session is being used to compute the *policyHash* and not for command authorization.  This setting modifies some policy commands and prevents session from being used to authorize a command. |
| #TPM\_RC\_VALUE |  | response code when incorrect value is used |

## TPM\_CAP (Capabilities)

The TPM\_CAP values are used in TPM2\_GetCapability() to select the type of the value to be returned. The format of the response varies according to the type of the value.

Table 22 — Definition of (UINT32) TPM\_CAP Constants

| Capability Name | Value | Property Type | Return Type |
| --- | --- | --- | --- |
| TPM\_CAP\_FIRST | 0x00000000 |  |  |
| TPM\_CAP\_ALGS | 0x00000000 | TPM\_ALG\_ID(1) | TPML\_ALG\_PROPERTY |
| TPM\_CAP\_HANDLES | 0x00000001 | TPM\_HANDLE | TPML\_HANDLE |
| TPM\_CAP\_COMMANDS | 0x00000002 | TPM\_CC | TPML\_CCA |
| TPM\_CAP\_PP\_COMMANDS | 0x00000003 | TPM\_CC | TPML\_CC |
| TPM\_CAP\_AUDIT\_COMMANDS | 0x00000004 | TPM\_CC | TPML\_CC |
| TPM\_CAP\_PCRS | 0x00000005 | reserved | TPML\_PCR\_SELECTION |
| TPM\_CAP\_TPM\_PROPERTIES | 0x00000006 | TPM\_PT | TPML\_TAGGED\_TPM\_PROPERTY |
| TPM\_CAP\_PCR\_PROPERTIES | 0x00000007 | TPM\_PT\_PCR | TPML\_TAGGED\_PCR\_PROPERTY |
| TPM\_CAP\_ECC\_CURVES | 0x00000008 | TPM\_ECC\_CURVE(1) | TPML\_ECC\_CURVE |
| TPM\_CAP\_AUTH\_POLICIES | 0x00000009 | TPM\_HANDLE(2)(3) | TPML\_TAGGED\_POLICY |
| TPM\_CAP\_ACT | 0x0000000A | TPM\_HANDLE(2(4)) | TPML\_ACT\_DATA |
| TPM\_CAP\_LAST | 0x0000000A |  |  |
| TPM\_CAP\_VENDOR\_PROPERTY | 0x00000100 | manufacturer specific | manufacturer-specific values |
| #TPM\_RC\_VALUE |  |  |  |
| NOTES:  (1) The TPM\_ALG\_ID or TPM\_ECC\_CURVE is cast to a UINT32  (2) The TPM will return TPM\_RC\_VALUE if the handle does not reference the range for permanent handles.  (3) TPM\_CAP\_AUTH\_POLICIES was added in revision 01.32.  (4) TPM\_CAP\_ACT was added in revision 01.57. | | | |

## TPM\_PT (Property Tag)

The TPM\_PT constants are used in TPM2\_GetCapability(capability = TPM\_CAP\_TPM\_PROPERTIES) to indicate the property being selected or returned.

The values in the fixed group (PT\_FIXED) are not changeable through programmatic means other than a firmware update. The values in the variable group (PT\_VAR) may be changed with TPM commands but should be persistent over power cycles and only changed when indicated by the detailed actions code.

Table 23 — Definition of (UINT32) TPM\_PT Constants <IN/OUT, S>

| Capability Name | Value | Comments |
| --- | --- | --- |
| TPM\_PT\_NONE | 0x00000000 | indicates no property type |
| PT\_GROUP | 0x00000100 | The number of properties in each group.  NOTE The first group with any properties is group 1 (PT\_GROUP \* 1). Group 0 is reserved. |
| PT\_FIXED | PT\_GROUP \* 1 | the group of fixed properties returned as TPMS\_TAGGED\_PROPERTY  The values in this group are only changed due to a firmware change in the TPM. |
| TPM\_PT\_FAMILY\_INDICATOR | PT\_FIXED + 0 | a 4-octet character string containing the TPM Family value (TPM\_SPEC\_FAMILY) |
| TPM\_PT\_LEVEL | PT\_FIXED + 1 | the level of the specification  NOTE 1 For this specification, the level is zero.  NOTE 2 The level is on the title page of the specification. |
| TPM\_PT\_REVISION | PT\_FIXED + 2 | the specification Revision times 100  EXAMPLE Revision 01.01 would have a value of 101.  NOTE The Revision value is on the title page of the specification. |
| TPM\_PT\_DAY\_OF\_YEAR | PT\_FIXED + 3 | the specification day of year using TCG calendar  EXAMPLE November 15, 2010, has a day of year value of 319 (00 00 01 3F16).  NOTE The specification date is on the title page of the specification or errata (see 6.1). |
| TPM\_PT\_YEAR | PT\_FIXED + 4 | the specification year using the CE  EXAMPLE The year 2010 has a value of 00 00 07 DA16.  NOTE The specification date is on the title page of the specification or errata (see 6.1). |
| TPM\_PT\_MANUFACTURER | PT\_FIXED + 5 | the vendor ID unique to each TPM manufacturer |
| TPM\_PT\_VENDOR\_STRING\_1 | PT\_FIXED + 6 | the first four characters of the vendor ID string  NOTE When the vendor string is fewer than 16 octets, the additional property values do not have to be present. A vendor string of 4 octets can be represented in one 32-bit value and no null terminating character is required. |
| TPM\_PT\_VENDOR\_STRING\_2 | PT\_FIXED + 7 | the second four characters of the vendor ID string |
| TPM\_PT\_VENDOR\_STRING\_3 | PT\_FIXED + 8 | the third four characters of the vendor ID string |
| TPM\_PT\_VENDOR\_STRING\_4 | PT\_FIXED + 9 | the fourth four characters of the vendor ID sting |
| TPM\_PT\_VENDOR\_TPM\_TYPE | PT\_FIXED + 10 | vendor-defined value indicating the TPM model |
| TPM\_PT\_FIRMWARE\_VERSION\_1 | PT\_FIXED + 11 | the most-significant 32 bits of a TPM vendor-specific value indicating the version number of the firmware. See 10.12.2 and 10.12.12. |
| TPM\_PT\_FIRMWARE\_VERSION\_2 | PT\_FIXED + 12 | the least-significant 32 bits of a TPM vendor-specific value indicating the version number of the firmware. See 10.12.2 and 10.12.12. |
| TPM\_PT\_INPUT\_BUFFER | PT\_FIXED + 13 | the maximum size of a parameter (typically, a TPM2B\_MAX\_BUFFER) |
| TPM\_PT\_HR\_TRANSIENT\_MIN | PT\_FIXED + 14 | the minimum number of transient objects that can be held in TPM RAM  NOTE This minimum shall be no less than the minimum value required by the platform-specific specification to which the TPM is built. |
| TPM\_PT\_HR\_PERSISTENT\_MIN | PT\_FIXED + 15 | the minimum number of persistent objects that can be held in TPM NV memory  NOTE This minimum shall be no less than the minimum value required by the platform-specific specification to which the TPM is built. |
| TPM\_PT\_HR\_LOADED\_MIN | PT\_FIXED + 16 | the minimum number of authorization sessions that can be held in TPM RAM  NOTE This minimum shall be no less than the minimum value required by the platform-specific specification to which the TPM is built. |
| TPM\_PT\_ACTIVE\_SESSIONS\_MAX | PT\_FIXED + 17 | the number of authorization sessions that may be active at a time  A session is active when it has a context associated with its handle. The context may either be in TPM RAM or be context saved.  NOTE This value shall be no less than the minimum value required by the platform-specific specification to which the TPM is built. |
| TPM\_PT\_PCR\_COUNT | PT\_FIXED + 18 | the number of PCR implemented  NOTE This number is determined by the defined attributes, not the number of PCR that are populated. |
| TPM\_PT\_PCR\_SELECT\_MIN | PT\_FIXED + 19 | the minimum number of octets in a TPMS\_PCR\_SELECT.*sizeOfSelect*  NOTE This value is not determined by the number of PCR implemented but by the number of PCR required by the platform-specific specification with which the TPM is compliant or by the implementer if not adhering to a platform-specific specification. |
| TPM\_PT\_CONTEXT\_GAP\_MAX | PT\_FIXED + 20 | the maximum allowed difference (unsigned) between the *contextID* values of two saved session contexts  This value shall be 2n-1, where n is at least 16. |
|  | PT\_FIXED + 21 | skipped |
| TPM\_PT\_NV\_COUNTERS\_MAX | PT\_FIXED + 22 | the maximum number of NV Indexes that are allowed to have the TPM\_NT\_COUNTER attribute  NOTE 1 It is allowed for this value to be larger than the number of NV Indexes that can be defined. This would be indicative of a TPM implementation that did not use different implementation technology for different NV Index types.  NOTE 2 The value zero indicates that there is no fixed maximum. The number of counter indexes is determined by the available NV memory pool. |
| TPM\_PT\_NV\_INDEX\_MAX | PT\_FIXED + 23 | the maximum size of an NV Index data area |
| TPM\_PT\_MEMORY | PT\_FIXED + 24 | a TPMA\_MEMORY indicating the memory management method for the TPM |
| TPM\_PT\_CLOCK\_UPDATE | PT\_FIXED + 25 | interval, in milliseconds, between updates to the copy of TPMS\_CLOCK\_INFO.*clock* in NV |
| TPM\_PT\_CONTEXT\_HASH | PT\_FIXED + 26 | the algorithm used for the integrity HMAC on saved contexts and for hashing the *fuData* of TPM2\_FirmwareRead() |
| TPM\_PT\_CONTEXT\_SYM | PT\_FIXED + 27 | TPM\_ALG\_ID, the algorithm used for encryption of saved contexts |
| TPM\_PT\_CONTEXT\_SYM\_SIZE | PT\_FIXED + 28 | TPM\_KEY\_BITS, the size of the key used for encryption of saved contexts |
| TPM\_PT\_ORDERLY\_COUNT | PT\_FIXED + 29 | the modulus - 1 of the count for NV update of an orderly counter  The returned value is MAX\_ORDERLY\_COUNT.  This will have a value of 2N – 1 where 1 ≤ N ≤ 32  NOTE 1 An “orderly counter” is an NV Index with an TPM\_NT of TPM\_NV\_COUNTER and TPMA\_NV\_ORDERLY SET.  NOTE 2 When the low-order bits of a counter equal this value, an NV write occurs on the next increment. |
| TPM\_PT\_MAX\_COMMAND\_SIZE | PT\_FIXED + 30 | the maximum value for *commandSize* in a command |
| TPM\_PT\_MAX\_RESPONSE\_SIZE | PT\_FIXED + 31 | the maximum value for *responseSize* in a response |
| TPM\_PT\_MAX\_DIGEST | PT\_FIXED + 32 | the maximum size of a digest that can be produced by the TPM |
| TPM\_PT\_MAX\_OBJECT\_CONTEXT | PT\_FIXED + 33 | the maximum size of an object context that will be returned by TPM2\_ContextSave |
| TPM\_PT\_MAX\_SESSION\_CONTEXT | PT\_FIXED + 34 | the maximum size of a session context that will be returned by TPM2\_ContextSave |
| TPM\_PT\_PS\_FAMILY\_INDICATOR | PT\_FIXED + 35 | platform-specific family (a TPM\_PS value)(see Table 25)  NOTE The platform-specific values for the TPM\_PT\_PS parameters are in the relevant platform-specific specification. In the reference implementation, all of these values are 0. |
| TPM\_PT\_PS\_LEVEL | PT\_FIXED + 36 | the level of the platform-specific specification |
| TPM\_PT\_PS\_REVISION | PT\_FIXED + 37 | a platform specific value |
| TPM\_PT\_PS\_DAY\_OF\_YEAR | PT\_FIXED + 38 | the platform-specific TPM specification day of year using TCG calendar  EXAMPLE November 15, 2010, has a day of year value of 319 (00 00 01 3F16). |
| TPM\_PT\_PS\_YEAR | PT\_FIXED + 39 | the platform-specific TPM specification year using the CE  EXAMPLE The year 2010 has a value of 00 00 07 DA16. |
| TPM\_PT\_SPLIT\_MAX | PT\_FIXED + 40 | the number of split signing operations supported by the TPM |
| TPM\_PT\_TOTAL\_COMMANDS | PT\_FIXED + 41 | total number of commands implemented in the TPM |
| TPM\_PT\_LIBRARY\_COMMANDS | PT\_FIXED + 42 | number of commands from the TPM library that are implemented |
| TPM\_PT\_VENDOR\_COMMANDS | PT\_FIXED + 43 | number of vendor commands that are implemented |
| TPM\_PT\_NV\_BUFFER\_MAX | PT\_FIXED + 44 | the maximum data size in one NV write, NV read, NV extend, or NV certify command |
| TPM\_PT\_MODES | PT\_FIXED + 45 | a TPMA\_MODES value, indicating that the TPM is designed for these modes. |
| TPM\_PT\_MAX\_CAP\_BUFFER | PT\_FIXED + 46 | the maximum size of a TPMS\_CAPABILITY\_DATA structure returned in TPM2\_GetCapability(). |
|  |  | Intentionally left empty |
| PT\_VAR | PT\_GROUP \* 2 | the group of variable properties returned as TPMS\_TAGGED\_PROPERTY  The properties in this group change because of a Protected Capability other than a firmware update. The values are not necessarily persistent across all power transitions. |
| TPM\_PT\_PERMANENT | PT\_VAR + 0 | TPMA\_PERMANENT |
| TPM\_PT\_STARTUP\_CLEAR | PT\_VAR + 1 | TPMA\_STARTUP\_CLEAR |
| TPM\_PT\_HR\_NV\_INDEX | PT\_VAR + 2 | the number of NV Indexes currently defined |
| TPM\_PT\_HR\_LOADED | PT\_VAR + 3 | the number of authorization sessions currently loaded into TPM RAM |
| TPM\_PT\_HR\_LOADED\_AVAIL | PT\_VAR + 4 | the number of additional authorization sessions, of any type, that could be loaded into TPM RAM  This value is an estimate. If this value is at least 1, then at least one authorization session of any type may be loaded. Any command that changes the RAM memory allocation can make this estimate invalid.  NOTE A valid implementation may return 1 even if more than one authorization session would fit into RAM. |
| TPM\_PT\_HR\_ACTIVE | PT\_VAR + 5 | the number of active authorization sessions currently being tracked by the TPM  This is the sum of the loaded and saved sessions. |
| TPM\_PT\_HR\_ACTIVE\_AVAIL | PT\_VAR + 6 | the number of additional authorization sessions, of any type, that could be created  This value is an estimate. If this value is at least 1, then at least one authorization session of any type may be created. Any command that changes the RAM memory allocation can make this estimate invalid.  NOTE A valid implementation may return 1 even if more than one authorization session could be created. |
| TPM\_PT\_HR\_TRANSIENT\_AVAIL | PT\_VAR + 7 | estimate of the number of additional transient objects that could be loaded into TPM RAM  This value is an estimate. If this value is at least 1, then at least one object of any type may be loaded. Any command that changes the memory allocation can make this estimate invalid.  NOTE A valid implementation may return 1 even if more than one transient object would fit into RAM. |
| TPM\_PT\_HR\_PERSISTENT | PT\_VAR + 8 | the number of persistent objects currently loaded into TPM NV memory |
| TPM\_PT\_HR\_PERSISTENT\_AVAIL | PT\_VAR + 9 | the number of additional persistent objects that could be loaded into NV memory  This value is an estimate. If this value is at least 1, then at least one object of any type may be made persistent. Any command that changes the NV memory allocation can make this estimate invalid.  NOTE A valid implementation may return 1 even if more than one persistent object would fit into NV memory. |
| TPM\_PT\_NV\_COUNTERS | PT\_VAR + 10 | the number of defined NV Indexes that have NV the TPM\_NT\_COUNTER attribute |
| TPM\_PT\_NV\_COUNTERS\_AVAIL | PT\_VAR + 11 | the number of additional NV Indexes that can be defined with their TPM\_NT of TPM\_NV\_COUNTER and the TPMA\_NV\_ORDERLY attribute SET  This value is an estimate. If this value is at least 1, then at least one NV Index may be created with a TPM\_NT of TPM\_NV\_COUNTER and the TPMA\_NV\_ORDERLY attributes. Any command that changes the NV memory allocation can make this estimate invalid.  NOTE A valid implementation may return 1 even if more than one NV counter could be defined. |
| TPM\_PT\_ALGORITHM\_SET | PT\_VAR + 12 | code that limits the algorithms that may be used with the TPM |
| TPM\_PT\_LOADED\_CURVES | PT\_VAR + 13 | the number of loaded ECC curves |
| TPM\_PT\_LOCKOUT\_COUNTER | PT\_VAR + 14 | the current value of the lockout counter (*failedTries*) |
| TPM\_PT\_MAX\_AUTH\_FAIL | PT\_VAR + 15 | the number of authorization failures before DA lockout is invoked |
| TPM\_PT\_LOCKOUT\_INTERVAL | PT\_VAR + 16 | the number of seconds before the value reported by TPM\_PT\_LOCKOUT\_COUNTER is decremented |
| TPM\_PT\_LOCKOUT\_RECOVERY | PT\_VAR + 17 | the number of seconds after a lockoutAuth failure before use of lockoutAuth may be attempted again |
| TPM\_PT\_NV\_WRITE\_RECOVERY | PT\_VAR + 18 | number of milliseconds before the TPM will accept another command that will modify NV  This value is an approximation and may go up or down over time. |
| TPM\_PT\_AUDIT\_COUNTER\_0 | PT\_VAR + 19 | the high-order 32 bits of the command audit counter |
| TPM\_PT\_AUDIT\_COUNTER\_1 | PT\_VAR + 20 | the low-order 32 bits of the command audit counter |

## TPM\_PT\_PCR (PCR Property Tag)

The TPM\_PT\_PCR constants are used in TPM2\_GetCapability() to indicate the property being selected or returned. The PCR properties can be read when *capability* == TPM\_CAP\_PCR\_PROPERTIES. If there is no property that corresponds to the value of *property*, the next higher value is returned, if it exists.

Table 24 — Definition of (UINT32) TPM\_PT\_PCR Constants <IN/OUT, S>

| Capability Name | Value | Comments |
| --- | --- | --- |
| TPM\_PT\_PCR\_FIRST | 0x00000000 | bottom of the range of TPM\_PT\_PCR properties |
| TPM\_PT\_PCR\_SAVE | 0x00000000 | a SET bit in the TPMS\_PCR\_SELECT indicates that the PCR is saved and restored by TPM\_SU\_STATE |
| TPM\_PT\_PCR\_EXTEND\_L0 | 0x00000001 | a SET bit in the TPMS\_PCR\_SELECT indicates that the PCR may be extended from locality 0  This property is only present if a locality other than 0 is implemented. |
| TPM\_PT\_PCR\_RESET\_L0 | 0x00000002 | a SET bit in the TPMS\_PCR\_SELECT indicates that the PCR may be reset by TPM2\_PCR\_Reset() from locality 0 |
| TPM\_PT\_PCR\_EXTEND\_L1 | 0x00000003 | a SET bit in the TPMS\_PCR\_SELECT indicates that the PCR may be extended from locality 1  This property is only present if locality 1 is implemented. |
| TPM\_PT\_PCR\_RESET\_L1 | 0x00000004 | a SET bit in the TPMS\_PCR\_SELECT indicates that the PCR may be reset by TPM2\_PCR\_Reset() from locality 1  This property is only present if locality 1 is implemented. |
| TPM\_PT\_PCR\_EXTEND\_L2 | 0x00000005 | a SET bit in the TPMS\_PCR\_SELECT indicates that the PCR may be extended from locality 2  This property is only present if localities 1 and 2 are implemented. |
| TPM\_PT\_PCR\_RESET\_L2 | 0x00000006 | a SET bit in the TPMS\_PCR\_SELECT indicates that the PCR may be reset by TPM2\_PCR\_Reset() from locality 2  This property is only present if localities 1 and 2 are implemented. |
| TPM\_PT\_PCR\_EXTEND\_L3 | 0x00000007 | a SET bit in the TPMS\_PCR\_SELECT indicates that the PCR may be extended from locality 3  This property is only present if localities 1, 2, and 3 are implemented. |
| TPM\_PT\_PCR\_RESET\_L3 | 0x00000008 | a SET bit in the TPMS\_PCR\_SELECT indicates that the PCR may be reset by TPM2\_PCR\_Reset() from locality 3  This property is only present if localities 1, 2, and 3 are implemented. |
| TPM\_PT\_PCR\_EXTEND\_L4 | 0x00000009 | a SET bit in the TPMS\_PCR\_SELECT indicates that the PCR may be extended from locality 4  This property is only present if localities 1, 2, 3, and 4 are implemented. |
| TPM\_PT\_PCR\_RESET\_L4 | 0x0000000A | a SET bit in the TPMS\_PCR\_SELECT indicates that the PCR may be reset by TPM2\_PCR\_Reset() from locality 4  This property is only present if localities 1, 2, 3, and 4 are implemented. |
| reserved | 0x0000000B – 0x00000010 | the values in this range are reserved  They correspond to values that may be used to describe attributes associated with the extended localities (32-255).synthesize additional software localities. The meaning of these properties need not be the same as the meaning for the Extend and Reset properties above. |
| TPM\_PT\_PCR\_NO\_INCREMENT | 0x00000011 | a SET bit in the TPMS\_PCR\_SELECT indicates that modifications to this PCR (reset or Extend) will not increment the *pcrUpdateCounter* |
| TPM\_PT\_PCR\_DRTM\_RESET | 0x00000012 | a SET bit in the TPMS\_PCR\_SELECT indicates that the PCR is reset by a D-RTM event  These PCR are reset to -1 on TPM2\_Startup() and reset to 0 on a \_TPM\_Hash\_End event following a \_TPM\_Hash\_Start event. |
| TPM\_PT\_PCR\_POLICY | 0x00000013 | a SET bit in the TPMS\_PCR\_SELECT indicates that the PCR is controlled by policy  This property is only present if the TPM supports policy control of a PCR. |
| TPM\_PT\_PCR\_AUTH | 0x00000014 | a SET bit in the TPMS\_PCR\_SELECT indicates that the PCR is controlled by an authorization value  This property is only present if the TPM supports authorization control of a PCR. |
| reserved | 0x00000015 | reserved for the next (2nd) TPM\_PT\_PCR\_POLICY set |
| reserved | 0x00000016 | reserved for the next (2nd) TPM\_PT\_PCR\_AUTH set |
| reserved | 0x00000017 – 0x00000210 | reserved for the 2nd through 255th TPM\_PT\_PCR\_POLICY and TPM\_PT\_PCR\_AUTH values |
| reserved | 0x00000211 | reserved to the 256th, and highest allowed, TPM\_PT\_PCR\_POLICY set |
| reserved | 0x00000212 | reserved to the 256th, and highest allowed, TPM\_PT\_PCR\_AUTH set |
| reserved | 0x00000213 | new PCR property values may be assigned starting with this value |
| TPM\_PT\_PCR\_LAST | 0x00000014 | top of the range of TPM\_PT\_PCR properties of the implementation  If the TPM receives a request for a PCR property with a value larger than this, the TPM will return a zero length list and set the *moreData* parameter to NO.  NOTE This is an implementation-specific value. The value shown reflects the reference code implementation. |

## TPM\_PS (Platform Specific)

The platform values in Table 25 are used for the TPM\_PT\_PS\_FAMILY\_INDICATOR.

Table 25 is an informative example of a TPM\_PS constants table in the TCG Registry of Reserved TPM 2.0 Handles and Localities. It is provided for illustrative purposes only.

NOTE Values below six (6) have the same values as the purview assignments in TPM 1.2.

Table 25 — Definition of (UINT32) TPM\_PS Constants <OUT>

| Capability Name | Value | Comments |
| --- | --- | --- |
| TPM\_PS\_MAIN | 0x00000000 | not platform specific |
| TPM\_PS\_PC | 0x00000001 | PC Client |
| TPM\_PS\_PDA | 0x00000002 | PDA (includes all mobile devices that are not specifically cell phones) |
| TPM\_PS\_CELL\_PHONE | 0x00000003 | Cell Phone |
| TPM\_PS\_SERVER | 0x00000004 | Server WG |
| TPM\_PS\_PERIPHERAL | 0x00000005 | Peripheral WG |
| TPM\_PS\_TSS | 0x00000006 | TSS WG (deprecated) |
| TPM\_PS\_STORAGE | 0x00000007 | Storage WG |
| TPM\_PS\_AUTHENTICATION | 0x00000008 | Authentication WG |
| TPM\_PS\_EMBEDDED | 0x00000009 | Embedded WG |
| TPM\_PS\_HARDCOPY | 0x0000000A | Hardcopy WG |
| TPM\_PS\_INFRASTRUCTURE | 0x0000000B | Infrastructure WG (deprecated) |
| TPM\_PS\_VIRTUALIZATION | 0x0000000C | Virtualization WG |
| TPM\_PS\_TNC | 0x0000000D | Trusted Network Connect WG (deprecated) |
| TPM\_PS\_MULTI\_TENANT | 0x0000000E | Multi-tenant WG (deprecated) |
| TPM\_PS\_TC | 0x0000000F | Technical Committee (deprecated) |

# Handles

## Introduction

Handles are 32-bit values used to reference shielded locations of various types within the TPM.

Table 26 — Definition of Types for Handles

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| UINT32 | TPM\_HANDLE |  |

Handles may refer to objects (keys or data blobs), authorization sessions (HMAC and policy), NV Indexes, permanent TPM locations, and PCR.

## TPM\_HT (Handle Types)

The 32-bit handle space is divided into 256 regions of equal size with 224 values in each. Each of these ranges represents a handle type.

The type of the entity is indicated by the MSO of its handle. The values for the MSO and the entity referenced are shown in Table 27.

Table 27 — Definition of (UINT8) TPM\_HT Constants <S>

| Name | Value | Comments |
| --- | --- | --- |
| TPM\_HT\_PCR | 0x00 | **PCR** – consecutive numbers, starting at 0, that reference the PCR registers  A platform-specific specification will set the minimum number of PCR and an implementation may have more. |
| TPM\_HT\_NV\_INDEX | 0x01 | **NV Index** – assigned by the caller |
| TPM\_HT\_HMAC\_SESSION | 0x02 | **HMAC Authorization Session** – assigned by the TPM when the session is created |
| TPM\_HT\_LOADED\_SESSION | 0x02 | **Loaded Authorization Session** – used only in the context of TPM2\_GetCapability  This type references both loaded HMAC and loaded policy authorization sessions. |
| TPM\_HT\_POLICY\_SESSION | 0x03 | **Policy Authorization Session** – assigned by the TPM when the session is created |
| TPM\_HT\_SAVED\_SESSION | 0x03 | **Saved Authorization Session** – used only in the context of TPM2\_GetCapability  This type references saved authorization session contexts for which the TPM is maintaining tracking information. |
| TPM\_HT\_PERMANENT | 0x40 | **Permanent Values** – assigned by this specification in Table 28 |
| TPM\_HT\_TRANSIENT | 0x80 | **Transient Objects** – assigned by the TPM when an object is loaded into transient-object memory or when a persistent object is converted to a transient object |
| TPM\_HT\_PERSISTENT | 0x81 | **Persistent Objects** – assigned by the TPM when a loaded transient object is made persistent |
| TPM\_HT\_AC | 0x90 | **Attached Component** – handle for an Attached Component. |

When a transient object is loaded, the TPM shall assign a handle with an MSO of TPM\_HT\_TRANSIENT. The object may be assigned a different handle each time it is loaded. The TPM shall ensure that handles assigned to transient objects are unique and assigned to only one transient object at a time.

EXAMPLE 1 If a TPM is only able to hold 4 transient objects in internal memory, it might choose to assign handles to those objects with the values 80 00 00 0016 – 80 00 00 0316.

When a transient object is converted to a persistent object (TPM2\_EvictControl()), the TPM shall validate that the handle provided by the caller has an MSO of TPM\_HT\_PERSISTENT and that the handle is not already assigned to a persistent object.

A handle is assigned to a session when the session is started. The handle shall have an MSO equal to TPM\_HT\_SESSION and remain associated with that session until the session is closed or flushed. The TPM shall ensure that a session handle is only associated with one session at a time. When the session is loaded into the TPM using TPM2\_LoadContext(), it will have the same handle each time it is loaded.

EXAMPLE 2 If a TPM is only able to track 64 active sessions at a time, it could number those sessions using the values xx 00 01 0016 – xx 00 01 3F16 where xx is either 0216 or 0316 depending on the session type.

## Persistent Handle Sub-ranges

Persistent handles are assigned by the caller of TPM2\_EvictControl(). Owner Authorization or Platform Authorization is required to authorize allocation of space for a persistent object. These entities are given separate ranges of persistent handles so that they do not have to allocate from a common range of handles.

NOTE While this “namespace” allocation of the handle ranges could have been handled by convention, TPM enforcement is used to prevent errors by the OS or malicious software from affecting the platform’s use of the NV memory.

The Owner is allocated persistent handles in the range of 81 00 00 0016 to 81 7F FF FF16 inclusive and the TPM will return an error if Owner Authorization is used to attempt to assign a persistent handle outside of this range.

## TPM\_RH (Permanent Handles)

Table 28 lists the architecturally defined handles that cannot be changed. The handles include authorization handles, and special handles.

Table 28 — Definition of (TPM\_HANDLE) TPM\_RH Constants <S>

| Name | Value | Type | Comments |
| --- | --- | --- | --- |
| TPM\_RH\_FIRST | 0x40000000 | R |  |
| TPM\_RH\_SRK | 0x40000000 | R | not used1 |
| TPM\_RH\_OWNER | 0x40000001 | K, A, P | handle references the Storage Primary Seed (SPS), the *ownerAuth*, and the *ownerPolicy* |
| TPM\_RH\_REVOKE | 0x40000002 | R | not used1 |
| TPM\_RH\_TRANSPORT | 0x40000003 | R | not used1 |
| TPM\_RH\_OPERATOR | 0x40000004 | R | not used1 |
| TPM\_RH\_ADMIN | 0x40000005 | R | not used1 |
| TPM\_RH\_EK | 0x40000006 | R | not used1 |
| TPM\_RH\_NULL | 0x40000007 | K, A, P | a handle associated with the null hierarchy, an EmptyAuth *authValue*, and an Empty Policy *authPolicy*. |
| TPM\_RH\_UNASSIGNED | 0x40000008 | R | value reserved to the TPM to indicate a handle location that has not been initialized or assigned |
| TPM\_RS\_PW | 0x40000009 | S | authorization value used to indicate a password authorization session |
| TPM\_RH\_LOCKOUT | 0x4000000A | A | references the authorization associated with the dictionary attack lockout reset |
| TPM\_RH\_ENDORSEMENT | 0x4000000B | K, A, P | references the Endorsement Primary Seed (EPS), *endorsementAuth*, and *endorsementPolicy* |
| TPM\_RH\_PLATFORM | 0x4000000C | K, A, P | references the Platform Primary Seed (PPS), *platformAuth*, and *platformPolicy* |
| TPM\_RH\_PLATFORM\_NV | 0x4000000D | C | for phEnableNV |
| TPM\_RH\_AUTH\_00 | 0x40000010 | A | Start of a range of authorization values that are vendor-specific. A TPM may support any of the values in this range as are needed for vendor-specific purposes.  Disabled if ehEnable is CLEAR.  NOTE “Any” includes “none”. |
| TPM\_RH\_AUTH\_FF | 0x4000010F | A | End of the range of vendor-specific authorization values. |
| TPM\_RH\_ACT\_0 | 0x40000110 | A,P | Start of the range of authenticated timers |
| TPM\_RH\_ACT\_F | 0x4000011F | A,P | End of the range of authenticated timers |
| TPM\_RH\_LAST | 0x4000011F | R | the top of the reserved handle area  This is set to allow TPM2\_GetCapability() to know where to stop. It may vary as implementations add to the permanent handle area. |
| Type definitions:  **R** – a reserved value  **K** – a Primary Seed  **A** – an authorization value  **P** – a policy value  **S** – a session handle  **C** - a control  Note 1 The handle is only used in a TPM that is compatible with a previous version of this specification. It is not used in any command defined in this version of the specification. | | | |

## TPM\_HC (Handle Value Constants)

The definitions in Table 29 are used to define many of the interface data types.

These values, that indicate ranges, are informative and may be changed by an implementation. The TPM will always return the correct handle type as described in 7.2 Table 27:

* HMAC\_SESSION\_FIRST—HMAC\_SESSION\_LAST,
* LOADED\_SESSION\_FIRST—LOADED\_SESSION\_LAST,
* POLICY\_SESSION\_FIRST—POLICY\_SESSION\_LAST,
* TRANSIENT\_FIRST—TRANSIENT\_LAST,
* ACTIVE\_SESSION\_FIRST—ACTIVE\_SESSION\_LAST,
* PCR\_FIRST—PCR\_LAST

These values are input by the caller. The TPM implementation should support the entire range:

* PERSISTENT\_FIRST—PERSISTENT\_LAST,
* PLATFORM\_PERSISTENT—PLATFORM\_PERSISTENT+0x007FFFFF,
* NV\_INDEX\_FIRST—NV\_INDEX\_LAST,
* PERMANENT\_FIRST—PERMANENT\_LAST

NOTE PCR0 is architecturally defined to have a handle value of 0.

For the reference implementation, the handle range for sessions starts at the lowest allowed value for a session handle. The highest value for a session handle is determined by how many active sessions are allowed by the implementation. The MSO of the session handle will be set according to the session type.

A similar approach is used for transient objects with the first assigned handle at the bottom of the range defined by TPM\_HT\_TRANSIENT and the top of the range determined by the implementation-dependent value of MAX\_LOADED\_OBJECTS.

The first assigned handle for evict objects is also at the bottom of the allowed range defined by TPM\_HT\_PERSISTENT and the top of the range determined by the implementation-dependent value of MAX\_EVICT\_OBJECTS.

NOTE The values in Table 29 are intended to facilitate the process of making the handle larger than 32 bits in the future. It is intended that HR\_MASK and HR\_SHIFT are the only values that need change to resize the handle space.

Table 29 — Definition of (TPM\_HANDLE) TPM\_HC Constants <S>

| Name | Value | Comments |
| --- | --- | --- |
| HR\_HANDLE\_MASK | 0x00FFFFFF | to mask off the HR |
| HR\_RANGE\_MASK | 0xFF000000 | to mask off the variable part |
| HR\_SHIFT | 24 |  |
| HR\_PCR | (TPM\_HT\_PCR << HR\_SHIFT) |  |
| HR\_HMAC\_SESSION | (TPM\_HT\_HMAC\_SESSION << HR\_SHIFT) |  |
| HR\_POLICY\_SESSION | (TPM\_HT\_POLICY\_SESSION << HR\_SHIFT) |  |
| HR\_TRANSIENT | (TPM\_HT\_TRANSIENT << HR\_SHIFT) |  |
| HR\_PERSISTENT | (TPM\_HT\_PERSISTENT << HR\_SHIFT) |  |
| HR\_NV\_INDEX | (TPM\_HT\_NV\_INDEX << HR\_SHIFT) |  |
| HR\_PERMANENT | (TPM\_HT\_PERMANENT << HR\_SHIFT) |  |
| PCR\_FIRST | (HR\_PCR + 0) | first PCR |
| PCR\_LAST | (PCR\_FIRST + IMPLEMENTATION\_PCR-1) | last PCR |
| HMAC\_SESSION\_FIRST | (HR\_HMAC\_SESSION + 0) | first HMAC session |
| HMAC\_SESSION\_LAST | (HMAC\_SESSION\_FIRST+MAX\_ACTIVE\_SESSIONS-1) | last HMAC session |
| LOADED\_SESSION\_FIRST | HMAC\_SESSION\_FIRST | used in GetCapability |
| LOADED\_SESSION\_LAST | HMAC\_SESSION\_LAST | used in GetCapability |
| POLICY\_SESSION\_FIRST | (HR\_POLICY\_SESSION + 0) | first policy session |
| POLICY\_SESSION\_LAST | (POLICY\_SESSION\_FIRST + MAX\_ACTIVE\_SESSIONS-1) | last policy session |
| TRANSIENT\_FIRST | (HR\_TRANSIENT + 0) | first transient object |
| ACTIVE\_SESSION\_FIRST | POLICY\_SESSION\_FIRST | used in GetCapability |
| ACTIVE\_SESSION\_LAST | POLICY\_SESSION\_LAST | used in GetCapability |
| TRANSIENT\_LAST | (TRANSIENT\_FIRST+MAX\_LOADED\_OBJECTS-1) | last transient object |
| PERSISTENT\_FIRST | (HR\_PERSISTENT + 0) | first persistent object |
| PERSISTENT\_LAST | (PERSISTENT\_FIRST + 0x00FFFFFF) | last persistent object |
| PLATFORM\_PERSISTENT | (PERSISTENT\_FIRST + 0x00800000) | first platform persistent object |
| NV\_INDEX\_FIRST | (HR\_NV\_INDEX + 0) | first allowed NV Index |
| NV\_INDEX\_LAST | (NV\_INDEX\_FIRST + 0x00FFFFFF) | last allowed NV Index |
| PERMANENT\_FIRST | TPM\_RH\_FIRST |  |
| PERMANENT\_LAST | TPM\_RH\_LAST |  |
| HR\_NV\_AC | ((TPM\_HT\_NV\_INDEX << HR\_SHIFT) + 0xD00000) | AC aliased NV Index |
| NV\_AC\_FIRST | (HR\_NV\_AC + 0) | first NV Index aliased to Attached Component |
| NV\_AC\_LAST | (HR\_NV\_AC + 0x0000FFFF) | last NV Index aliased to Attached Component |
| HR\_AC | (TPM\_HT\_AC << HR\_SHIFT) | AC Handle |
| AC\_FIRST | (HR\_AC + 0) | first Attached Component |
| AC\_LAST | (HR\_AC + 0x0000FFFF) | last Attached Component |

# Attribute Structures

## Description

Attributes are expressed as bit fields of varying size. An attribute field structure may be 1, 2, or 4 octets in length.

The bit numbers for an attribute structure are assigned with the number 0 assigned to the least-significant bit of the structure and the highest number assigned to the most-significant bit of the structure.

The least significant bit is determined by treating the attribute structure as an integer. The least-significant bit would be the bit that is set when the value of the integer is 1.

When any reserved bit in an attribute is SET, the TPM shall return TPM\_RC\_RESERVED\_BITS. This response code is not shown in the tables for attributes.

## TPMA\_ALGORITHM

This structure defines the attributes of an algorithm.

Each algorithm has a fundamental attribute: *asymmetric*, *symmetric*, or *hash*. In some cases (e.g., TPM\_ALG\_RSA or TPM\_ALG\_AES), this is the only attribute.

A mode, method, or scheme may have an associated asymmetric, symmetric, or hash algorithm.

NOTE A hash algorithm that can be used directly is one that has only the *hash* attribute SET.

EXAMPLE A PCR bank or an object Name can only use an algorithm that has only the *hash* attribute SET.

Table 30 — Definition of (UINT32) TPMA\_ALGORITHM Bits

| Bit | Name | Definition |
| --- | --- | --- |
| 0 | asymmetric | **SET (1):** an asymmetric algorithm with public and private portions  **CLEAR (0):** not an asymmetric algorithm |
| 1 | symmetric | **SET (1):** a symmetric block cipher  **CLEAR (0):** not a symmetric block cipher |
| 2 | hash | **SET (1):** a hash algorithm  **CLEAR (0):** not a hash algorithm |
| 3 | object | **SET (1):** an algorithm that may be used as an object type  **CLEAR (0):** an algorithm that is not used as an object type |
| 7:4 | Reserved |  |
| 8 | signing | **SET (1):** a signing algorithm. The setting of *asymmetric*, *symmetric*, and *hash* will indicate the type of signing algorithm.  **CLEAR (0):** not a signing algorithm |
| 9 | encrypting | **SET (1):** an encryption/decryption algorithm. The setting of *asymmetric*, *symmetric*, and *hash* will indicate the type of encryption/decryption algorithm.  **CLEAR (0):** not an encryption/decryption algorithm |
| 10 | method | **SET (1):** a method such as a key derivative function (KDF)  **CLEAR (0):** not a method |
| 31:11 | Reserved |  |

## TPMA\_OBJECT (Object Attributes)

### Introduction

This attribute structure indicates an object’s use, its authorization types, and its relationship to other objects.

The state of the attributes is determined when the object is created and they are never changed by the TPM. Additionally, the setting of these structures is reflected in the integrity value of the private area of an object in order to allow the TPM to detect modifications of the Protected Object when stored off the TPM.

### Structure Definition

Table 31 — Definition of (UINT32) TPMA\_OBJECT Bits

| Bit | Name | Definition |
| --- | --- | --- |
| 0 | Reserved | shall be zero |
| 1 | fixedTPM | **SET (1):** The hierarchy of the object, as indicated by its Qualified Name, may not change.  **CLEAR (0):** The hierarchy of the object may change as a result of this object or an ancestor key being duplicated for use in another hierarchy.  NOTE *fixedTPM* does not indicate that key material resides on a single TPM (see *sensitiveDataOrigin*)*.* |
| 2 | stClear | **SET (1):** Previously saved contexts of this object may not be loaded after Startup(CLEAR).  **CLEAR (0):** Saved contexts of this object may be used after a Shutdown(STATE) and subsequent Startup(). |
| 3 | Reserved | shall be zero |
| 4 | fixedParent | **SET (1):** The parent of the object may not change.  **CLEAR (0):** The parent of the object may change as the result of a TPM2\_Duplicate() of the object. |
| 5 | sensitiveDataOrigin | **SET (1):** Indicates that, when the object was created with TPM2\_Create() or TPM2\_CreatePrimary(), the TPM generated all of the sensitive data other than the *authValue*.  **CLEAR (0):** A portion of the sensitive data, other than the *authValue*, was provided by the caller. |
| 6 | userWithAuth | **SET (1):** Approval of USER role actions with this object may be with an HMAC session or with a password using the *authValue* of the object or a policy session.  **CLEAR (0):** Approval of USER role actions with this object may only be done with a policy session. |
| 7 | adminWithPolicy | **SET (1):** Approval of ADMIN role actions with this object may only be done with a policy session.  **CLEAR (0):** Approval of ADMIN role actions with this object may be with an HMAC session or with a password using the *authValue* of the object or a policy session. |
| 9:8 | Reserved | shall be zero |
| 10 | noDA | **SET (1):** The object is not subject to dictionary attack protections.  **CLEAR (0):** The object is subject to dictionary attack protections. |
| 11 | encryptedDuplication | **SET (1):** If the object is duplicated, then *symmetricAlg* shall not be TPM\_ALG\_NULL and *newParentHandle* shall not be TPM\_RH\_NULL.  **CLEAR (0):** The object may be duplicated without an inner wrapper on the private portion of the object and the new parent may be TPM\_RH\_NULL. |
| 15:12 | Reserved | shall be zero |
| 16 | restricted | **SET (1):** Key usage is restricted to manipulate structures of known format; the parent of this key shall have *restricted* SET.  **CLEAR (0):** Key usage is not restricted to use on special formats. |
| 17 | decrypt | **SET (1):** The private portion of the key may be used to decrypt.  **CLEAR (0):** The private portion of the key may not be used to decrypt. |
| 18 | sign / encrypt | **SET (1):** For a symmetric cipher object, the private portion of the key may be used to encrypt. For other objects, the private portion of the key may be used to sign.  **CLEAR (0):** The private portion of the key may not be used to sign or encrypt. |
| 19 | x509sign | **SET (1):** An asymmetric key that may not be used to sign with TPM2\_Sign()  **CLEAR (0):** A key that may be used with TPM2\_Sign() if *sign* is SET  NOTE: This attribute only has significance if *sign* is SET. |
| 31:20 | Reserved | shall be zero |

### Attribute Descriptions

#### Introduction

The following remaining paragraphs in 8.3.3 describe the use and settings for each of the TPMA\_OBJECT attributes. The description includes checks that are performed on the *objectAttributes* when an object is created, when it is loaded, and when it is imported. In these descriptions:

**Creation** indicates settings for the *template* parameter in TPM2\_Create() or TPM2\_CreatePrimary()

**Load** indicates settings for the *inPublic* parameter in TPM2\_Load()

**Import** indicates settings for the *objectPublic* parameter in TPM2\_Import()

**External** indicates settings that apply to the *inPublic* parameter in TPM2\_LoadExternal() if both the public and sensitive portions of the object are loaded

NOTE For TPM2\_LoadExternal() when only the public portion of the object is loaded, the only attribute checks are the checks in the validation code following Table 31 and the reserved attributes check.

For any consistency error of attributes in TPMA\_OBJECT, the TPM shall return TPM\_RC\_ATTRIBUTES.

#### Bit[1] – *fixedTPM*

When SET, the object cannot be duplicated for use on a different TPM, either directly or indirectly and the Qualified Name of the object cannot change. When CLEAR, the object’s Qualified Name may change if the object or an ancestor is duplicated.

NOTE This attribute is the logical inverse of the migratable attribute in 1.2. That is, when this attribute is CLEAR, it is the equivalent to a 1.2 object with migratable SET.

**Creation** If *fixedTPM* is SET in the object's parent, then *fixedTPM* and *fixedParent* shall both be set to the same value in *template.* If *fixedTPM* is CLEAR in the parent, this attribute shall also be CLEAR in *template*.

NOTE For a Primary Object, the parent is considered to have *fixedTPM* SET.

**Load** If *fixedTPM* is SET in the object's parent, then *fixedTPM* and *fixedParent* shall both be set to the same value. If *fixedTPM* is CLEAR in the parent, this attribute shall also be CLEAR.

**Import** shall be CLEAR

**External** shall be CLEAR if both the public and sensitive portions are loaded or if fixedParent is CLEAR, otherwise may be SET or CLEAR

#### Bit[2] – *stClear*

If this attribute is SET, then saved contexts of this object will be invalidated on TPM2\_Startup(TPM\_SU\_CLEAR). If the attribute is CLEAR, then the TPM shall not invalidate the saved context if the TPM received TPM2\_Shutdown(TPM\_SU\_STATE). If the saved state is valid when checked at the next TPM2\_Startup(), then the TPM shall continue to be able to use the saved contexts.

**Creation** may be SET or CLEAR in template

**Load** may be SET or CLEAR

**Import** may be SET or CLEAR

**External** may be SET or CLEAR

#### Bit[4] – *fixedParent*

If this attribute is SET, the object’s parent may not be changed. That is, this object may not be the object of a TPM2\_Duplicate(). If this attribute is CLEAR, then this object may be the object of a TPM2\_Duplicate().

**Creation** may be SET or CLEAR in template

**Load** may be SET or CLEAR

**Import** shall be CLEAR

**External** shall be CLEAR if both the public and sensitive portions are loaded; otherwise it may be SET or CLEAR

#### Bit[5] – *sensitiveDataOrigin*

This attribute is SET for any key that was generated by TPM in TPM2\_Create() or TPM2\_CreatePrimary(). If CLEAR, it indicates that the sensitive part of the object (other than the *obfuscation* value) was provided by the caller.

NOTE 1 If the *fixedTPM* attribute is SET, then this attribute is authoritative and accurately reflects the source of the sensitive area data. If the *fixedTPM* attribute is CLEAR, then validation of this attribute requires evaluation of the properties of the ancestor keys.

**Creation** If *inSensitive.sensitive.data.*size is zero, then this attribute shall be SET in the template; otherwise, it shall be CLEAR in the template.

NOTE 2 The *inSensitive.sensitive.data.size* parameter is required to be zero for an asymmetric key so *sensitiveDataOrigin* is required to be SET.

NOTE 3 The *inSensitive.sensitive.data.size* parameter may not be zero for a data object so *sensitiveDataOrigin* is required to be CLEAR. A data object has *type* = TPM\_ALG\_KEYEDHASH and its *sign* and *decrypt* attributes are CLEAR.

**Load** may be SET or CLEAR

**Import** may be SET or CLEAR

**External** may be SET or CLEAR

#### Bit[6] – *userWithAuth*

If SET, authorization for operations that require USER role authorization may be given if the caller provides proof of knowledge of the *authValue* of the object with an HMAC authorization session or a password.

If this attribute is CLEAR, then HMAC or password authorizations may not be used for USER role authorizations.

NOTE 1 Regardless of the setting of this attribute, authorizations for operations that require USER role authorizations may be provided with a policy session that satisfies the object's *authPolicy*.

NOTE 2 Regardless of the setting of this attribute, the *authValue* may be referenced in a policy session or used to provide the *bind* value in TPM2\_StartAuthSession(). However, if *userWithAuth* is CLEAR, then the object may be used as the bind object in TPM2\_StartAuthSession() but the session cannot be used to authorize actions on the object. If this were allowed, then the *userWithAuth* control could be circumvented simply by using the object as the bind object.

**Creation** may be SET or CLEAR in template

**Load** may be SET or CLEAR

**Import** may be SET or CLEAR

**External** may be SET or CLEAR

#### Bit[7] – *adminWithPolicy*

If CLEAR, authorization for operations that require ADMIN role may be given if the caller provides proof of knowledge of the *authValue* of the object with an HMAC authorization session or a password.

If this attribute is SET, then then HMAC or password authorizations may not be used for ADMIN role authorizations.

NOTE 1 Regardless of the setting of this attribute, operations that require ADMIN role authorization may be provided by a policy session that satisfies the object's *authPolicy*.

NOTE 2 This attribute is similar to *userWithAuth* but the logic is a bit different. When *userWithAuth* is CLEAR, the *authValue* may not be used for USER mode authorizations. When *adminWithPolicy* is CLEAR, it means that the *authValue* may be used for ADMIN role. Policy may always be used regardless of the setting of *userWithAuth* or *adminWithPolicy*.

Actions that always require policy (TPM2\_Duplicate()) are not affected by the setting of this attribute.

**Creation** may be SET or CLEAR in *template*

**Load** may be SET or CLEAR

**Import** may be SET or CLEAR

**External** may be SET or CLEAR

#### Bit[10] – *noDA*

If SET, then authorization failures for the object do not affect the dictionary attack protection logic and authorization of the object is not blocked if the TPM is in lockout.

**Creation** may be SET or CLEAR in *template*

**Load** may be SET or CLEAR

**Import** may be SET or CLEAR

**External** may be SET or CLEAR

#### Bit[11] – *encryptedDuplication*

If SET, then when the object is duplicated, the sensitive portion of the object is required to be encrypted with an inner wrapper and the new parent shall be an asymmetric key and not TPM\_RH\_NULL

NOTE 1 Enforcement of these requirements in TPM2\_Duplicate() is by not allowing *symmetricAlg* to be TPM\_ALG\_NULL and not allowing *newParentHandle* to be TPM\_RH\_NULL.

This attribute shall not be SET in any object that has *fixedTPM* SET**.**

NOTE 2 This requirement means that *encryptedDuplication* may not be SET if the object cannot be directly or indirectly duplicated.

If an object's parent has *fixedTPM* SET, and the object is duplicable (*fixedParent* == CLEAR), then *encryptedDuplication* may be SET or CLEAR in the object.

NOTE 3 This allows the object at the boundary between duplicable and non-duplicable objects to have either setting.

If an object's parent has *fixedTPM* CLEAR, then the object is required to have the same setting of *encryptedDuplication* as its parent.

NOTE 4 This requirement forces all duplicable objects in a duplication group to have the same *encryptedDuplication* setting.

**Creation** shall be CLEAR if *fixedTPM* is SET. If *fixedTPM* is CLEAR, then this attribute shall have the same value as its parent unless *fixedTPM* is SET in the object's parent, in which case, it may be SET or CLEAR.

**Load** shall be CLEAR if *fixedTPM* is SET. If *fixedTPM* is CLEAR, then this attribute shall have the same value as its parent, unless *fixedTPM* is SET the parent, in which case, it may be SET or CLEAR.

**Import** if *fixedTPM* is SET in the object's new parent, then this attribute may be SET or CLEAR, otherwise, it shall have the same setting as the new parent.

**External** may be SET or CLEAR.

#### Bit[16] – *restricted*

This this attribute modifies the *decrypt* and *sign* attributes of an object.

NOTE A key with this object CLEAR may not be a parent for another object.

**Creation** shall be CLEAR in *template* if neither sign nor decrypt is SET in *template*.

**Load** shall be CLEAR if neither sign nor decrypt is SET in the object

**Import** may be SET or CLEAR

**External** shall be CLEAR

#### Bit[17] – *decrypt*

When SET, the private portion of this key can be used to decrypt an external blob. If *restricted* is SET, then the TPM will return an error if the external decrypted blob is not formatted as appropriate for the command.

NOTE 1 Since TPM-generated keys and sealed data will contain a hash and a structure tag, the TPM can ensure that it is not being used to improperly decrypt and return sensitive data that should not be returned. The only type of data that may be returned after decryption is a Sealed Data Object (a *keyedHash* object with *decrypt* and *sign* CLEAR).

When *restricted* is CLEAR, there are no restrictions on the use of the private portion of the key for decryption and the key may be used to decrypt and return any structure encrypted by the public portion of the key.

NOTE 2 A key with this attribute SET may be a parent for another object if *restricted* is SET and *sign* is CLEAR.

If *decrypt* is SET on an object with *type* set to TPM\_ALG\_KEYEDHASH, it indicates that the object is an XOR encryption key.

**Creation** may be SET or CLEAR in *template*

**Load** may be SET or CLEAR

**Import** may be SET or CLEAR

**External** may be SET or CLEAR

#### Bit[18] – *sign* / *encrypt*

When SET, the private portion of this key may be used to sign a digest if the key is an asymetric key or to encrypt a block of data if the key is a symmetric key. If *restricted* is SET, then the asymmetric key may only be used to sign a digest that was computed by the TPM. A restricted symmetric key may only be used to encrypt a data block. If a structure is generated by the TPM, it will begin with TPM\_GENERATED\_VALUE and the TPM may sign the digest of that structure. If the data is externally supplied and has TPM\_GENERATED\_VALUE as its first octets, then the TPM will not sign a digest of that data with a restricted signing key.

If *restricted* is CLEAR, then the key may be used to sign any digest or encrypt any data block, whether generated by the TPM or externally provided.

NOTE 1 Some asymmetric algorithms may not support both *sign* and *decrypt* being SET in the same key.

If *sign* is SET on an object with *type* set to TPM\_ALG\_KEYEDHASH, it indicates that the object is an HMAC key.

NOTE 2 A key with this attribute SET may not be a parent for another object.

**Creation** shall not be SET if *decrypt* and *restricted* are both SET

**Load** shall not be SET if *decrypt* and *restricted* are both SET

**Import** shall not be SET if *decrypt* and *restricted* are both SET

**External** shall not be SET if *decrypt* and *restricted* are both SET

#### Bit[19] – *x509sign*

When SET, the private portion of the asymmetric key may not be used as the signing key in TPM2\_Sign(). This restriction is to ensure that the only digest signed by this key is a digest of a strucure that is specific to the TPM or an x509 certiticate.

NOTE 1 This attribute does not limit the use of the key in any command other than TPM2\_Sign().

NOTE 2 This attribute was added in revision 01.53.

This attribute may not be SET if the object is not an asymmetric key or if *sign* is CLEAR.

**Creation** shall not be SET if *sign* is CLEAR or if the object is not an asymmetric key

**Load** shall not be SET if *sign* is CLEAR or if the object is not an asymmetric key

**Import** shall not be SET if *sign* is CLEAR or if the object is not an asymmetric key

**External** shall not be SET if *sign* is CLEAR or if the object is not an asymmetric key

## TPMA\_SESSION (Session Attributes)

This octet in each session is used to identify the session type, indicate its relationship to any handles in the command, and indicate its use in parameter encryption.

If a session is not being used for authorization, at least one of decrypt, encrypt, or audit must be SET.

In this revision, if *audit* is CLEAR, *auditExclusive* must be CLEAR in the command and will be CLEAR in the response. In a future, revision, this bit may have a different meaning if *audit* is CLEAR. See "Exclusive Audit Session" clause in TPM 2.0 Part 1.

In this revision, if *audit* is CLEAR, *auditReset* must be clear in the command and will be CLEAR in the response. In a future, revision, this bit may have a different meaning if *audit* is CLEAR.

*decrypt* may only be SET in one session per command. It may only be SET if the first parameter of the command is a sized buffer (TPM2B\_).

*encrypt* may only be SET in one session per command. It may only be SET if the first parameter of the response is a sized buffer (TPM2B\_).

*audit* may only be SET in one session per command or response.

Table 32 — Definition of (UINT8) TPMA\_SESSION Bits <IN/OUT>

| Bit | Name | Meaning |
| --- | --- | --- |
| 0 | continueSession | **SET (1):** In a command, this setting indicates that the session is to remain active after successful completion of the command. In a response, it indicates that the session is still active. If SET in the command, this attribute shall be SET in the response.  **CLEAR (0):** In a command, this setting indicates that the TPM should close the session and flush any related context when the command completes successfully. In a response, it indicates that the session is closed and the context is no longer active.  This attribute has no meaning for a password authorization and the TPM will allow any setting of the attribute in the command and SET the attribute in the response.  This attribute will only be CLEAR in one response for a logical session. If the attribute is CLEAR, the context associated with the session is no longer in use and the space is available. A session created after another session is ended may have the same handle but logically is not the same session.  This attribute has no effect if the command does not complete successfully. |
| 1 | auditExclusive | **SET (1):** In a command, this setting indicates that the command should only be executed if the session is exclusive at the start of the command. In a response, it indicates that the session is exclusive. This setting is only allowed if the *audit* attribute is SET (TPM\_RC\_ATTRIBUTES).  **CLEAR (0):** In a command, indicates that the session need not be exclusive at the start of the command. In a response, indicates that the session is not exclusive. |
| 2 | auditReset | **SET (1):** In a command, this setting indicates that the audit digest of the session should be initialized and the exclusive status of the session SET. This setting is only allowed if the *audit* attribute is SET (TPM\_RC\_ATTRIBUTES).  **CLEAR (0):** In a command, indicates that the audit digest should not be initialized.  This bit is always CLEAR in a response. |
| 4:3 | Reserved | shall be CLEAR |
| 5 | decrypt | **SET (1):** In a command, this setting indicates that the first parameter in the command is symmetrically encrypted using the parameter encryption scheme described in TPM 2.0 Part 1. The TPM will decrypt the parameter after performing any HMAC computations and before unmarshaling the parameter. In a response, the attribute is copied from the request but has no effect on the response.  **CLEAR (0):** Session not used for encryption.  For a password authorization, this attribute will be CLEAR in both the command and response.  This attribute may be SET in a session that is not associated with a command handle. Such a session is provided for purposes of encrypting a parameter and not for authorization.  This attribute may be SET in combination with any other session attributes. |
| 6 | encrypt | **SET (1):** In a command, this setting indicates that the TPM should use this session to encrypt the first parameter in the response. In a response, it indicates that the attribute was set in the command and that the TPM used the session to encrypt the first parameter in the response using the parameter encryption scheme described in TPM 2.0 Part 1.  **CLEAR (0):** Session not used for encryption.  For a password authorization, this attribute will be CLEAR in both the command and response.  This attribute may be SET in a session that is not associated with a command handle. Such a session is provided for purposes of encrypting a parameter and not for authorization. |
| 7 | audit | **SET (1):** In a command or response, this setting indicates that the session is for audit and that *auditExclusive* and *auditReset* have meaning. This session may also be used for authorization, encryption, or decryption. The *encrypted* and *encrypt* fields may be SET or CLEAR.  **CLEAR (0):** Session is not used for audit.  If SET in the command, then this attribute will be SET in the response. |

## TPMA\_LOCALITY (Locality Attribute)

In a TPMS\_CREATION\_DATA structure, this structure is used to indicate the locality of the command that created the object. No more than one of the locality attributes shall be set in the creation data.

When used in TPM2\_PolicyLocality(), this structure indicates which localities are approved by the policy. When a policy is started, all localities are allowed. If TPM2\_PolicyLocality() is executed, it indicates that the command may only be executed at specific localities. More than one locality may be selected.

EXAMPLE 1 TPM\_LOC\_TWO would indicate that only locality 2 is authorized.

EXAMPLE 2 TPM\_LOC\_ONE + TPM\_LOC\_TWO would indicate that locality 1 or 2 is authorized.

EXAMPLE 3 TPM\_LOC\_FOUR + TPM\_LOC\_THREE would indicate that localities 3 or 4 are authorized.

EXAMPLE 4 A value of 2116 would represent a locality of 33.

NOTE Locality values of 5 through 31 are not selectable.

If Extended is non-zero, then an extended locality is indicated and the TPMA\_LOCALITY contains an integer value.

Table 33 — Definition of (UINT8) TPMA\_LOCALITY Bits <IN/OUT>

|  |  |  |
| --- | --- | --- |
| Bit | Name | Definition |
| 0 | TPM\_LOC\_ZERO |  |
| 1 | TPM\_LOC\_ONE |  |
| 2 | TPM\_LOC\_TWO |  |
| 3 | TPM\_LOC\_THREE |  |
| 4 | TPM\_LOC\_FOUR |  |
| 7:5 | Extended | If any of these bits is set, an extended locality is indicated |

## TPMA\_PERMANENT

The attributes in this structure are persistent and are not changed as a result of \_TPM\_Init or any TPM2\_Startup(). Some of the attributes in this structure may change as the result of specific Protected Capabilities. This structure may be read using TPM2\_GetCapability(*capability* = TPM\_CAP\_TPM\_PROPERTIES, *property* = TPM\_PT\_PERMANENT).

Table 34 — Definition of (UINT32) TPMA\_PERMANENT Bits <OUT>

| Bit | Parameter | Description |
| --- | --- | --- |
| 0 | ownerAuthSet | **SET (1):** TPM2\_HierarchyChangeAuth() with *ownerAuth* has been executed since the last TPM2\_Clear().  **CLEAR (0):** o*wnerAuth* has not been changed since TPM2\_Clear(). |
| 1 | endorsementAuthSet | **SET (1):** TPM2\_HierarchyChangeAuth() with *endorsementAuth* has been executed since the last TPM2\_Clear().  **CLEAR (0):** endorsementAuth has not been changed since TPM2\_Clear(). |
| 2 | lockoutAuthSet | **SET (1):** TPM2\_HierarchyChangeAuth() with *lockoutAuth* has been executed since the last TPM2\_Clear().  **CLEAR (0):** *lockoutAuth* has not been changed since TPM2\_Clear(). |
| 7:3 | Reserved |  |
| 8 | disableClear | **SET (1):** TPM2\_Clear() is disabled.  **CLEAR (0):** TPM2\_Clear() is enabled.  NOTE See “TPM2\_ClearControl” in TPM 2.0 Part 3 for details on changing this attribute. |
| 9 | inLockout | **SET (1):** The TPM is in lockout, when *failedTries* is equal to *maxTries*. |
| 10 | tpmGeneratedEPS | **SET (1):** The EPS was created by the TPM.  **CLEAR (0):** The EPS was created outside of the TPM using a manufacturer-specific process. |
| 31:11 | Reserved |  |

## TPMA\_STARTUP\_CLEAR

This structure may be read using TPM2\_GetCapability(*capability* = TPM\_CAP\_TPM\_PROPERTIES, *property* = TPM\_PT\_STARTUP\_CLEAR).

*phEnable* is SET on any TPM2\_Startup. *shEnable*, *ehEnable*, and *phEnableNV* are SET on TPM Reset or TPM\_Restart and preserved by TPM Resume.

Some of attributes may be changed as the result of specific Protected Capabilities.

Table 35 — Definition of (UINT32) TPMA\_STARTUP\_CLEAR Bits <OUT>

| Bit | Parameter | Description |
| --- | --- | --- |
| 0 | phEnable | **SET (1):** The platform hierarchy is enabled and *platformAuth* or *platformPolicy* may be used for authorization.  **CLEAR (0):** *platformAuth* and *platformPolicy* may not be used for authorizations, and objects in the platform hierarchy, including persistent objects, cannot be used.  NOTE See “TPM2\_HierarchyControl” in TPM 2.0 Part 3 for details on changing this attribute. |
| 1 | shEnable | **SET (1):** The Storage hierarchy is enabled and *ownerAuth* or *ownerPolicy* may be used for authorization. NV indices defined using owner authorization are accessible.  **CLEAR (0):** *ownerAuth* and *ownerPolicy* may not be used for authorizations, and objects in the Storage hierarchy, persistent objects, and NV indices defined using owner authorization cannot be used.  NOTE See “TPM2\_HierarchyControl” in TPM 2.0 Part 3 for details on changing this attribute. |
| 2 | ehEnable | **SET (1):** The EPS hierarchy is enabled and Endorsement Authorizationmay be used to authorize commands.  **CLEAR (0):** Endorsement Authorizationmay not be used for authorizations, and objects in the endorsement hierarchy, including persistent objects, cannot be used.  NOTE See “TPM2\_HierarchyControl” in TPM 2.0 Part 3 for details on changing this attribute. |
| 3 | phEnableNV | **SET (1):** NV indices that have TPMA\_NV\_PLATFORMCREATE SET may be read or written. The platform can create define and undefine indices.  **CLEAR (0):** NV indices that have TPMA\_NV\_PLATFORMCREATE SET may not be read or written (TPM\_RC\_HANDLE). The platform cannot define (TPM\_RC\_HIERARCHY) or undefined (TPM\_RC\_HANDLE) indices.  NOTE See “TPM2\_HierarchyControl” in TPM 2.0 Part 3 for details on changing this attribute.  NOTE  read refers to these commands: TPM2\_NV\_Read, TPM2\_NV\_ReadPublic, TPM\_NV\_Certify, TPM2\_PolicyNV  write refers to these commands: TPM2\_NV\_Write, TPM2\_NV\_Increment, TPM2\_NV\_Extend, TPM2\_NV\_SetBits  NOTE The TPM must query the index TPMA\_NV\_PLATFORMCREATE attribute to determine whether phEnableNV is applicable. Since the TPM will return TPM\_RC\_HANDLE if the index does not exist, it also returns this error code if the index is disabled. Otherwise, the TPM would leak the existence of an index even when disabled. |
| 30:4 | Reserved | shall be zero |
| 31 | orderly | **SET (1):** The TPM received a TPM2\_Shutdown() and a matching TPM2\_Startup().  **CLEAR (0):** TPM2\_Startup(TPM\_SU\_CLEAR) was not preceded by a TPM2\_Shutdown() of any type.  NOTE A shutdown is orderly if the TPM receives a TPM2\_Shutdown() of any type followed by a TPM2\_Startup() of any type. However, the TPM will return an error if TPM2\_Startup(TPM\_SU\_STATE) was not preceded by TPM2\_Shutdown(TPM\_SU\_STATE). |

## TPMA\_MEMORY

This structure of this attribute is used to report the memory management method used by the TPM for transient objects and authorization sessions. This structure may be read using TPM2\_GetCapability(*capability* = TPM\_CAP\_TPM\_PROPERTIES, *property* = TPM\_PT\_MEMORY).

If the RAM memory is shared, then context save of a session may make it possible to load an additional transient object.

Table 36 — Definition of (UINT32) TPMA\_MEMORY Bits <Out>

|  |  |  |
| --- | --- | --- |
| Bit | Name | Definition |
| 0 | sharedRAM | **SET (1):** indicates that the RAM memory used for authorization session contexts is shared with the memory used for transient objects  **CLEAR (0):** indicates that the memory used for authorization sessions is not shared with memory used for transient objects |
| 1 | sharedNV | **SET (1):** indicates that the NV memory used for persistent objects is shared with the NV memory used for NV Index values  **CLEAR (0):** indicates that the persistent objects and NV Index values are allocated from separate sections of NV |
| 2 | objectCopiedToRam | **SET (1):** indicates that the TPM copies persistent objects to a transient-object slot in RAM when the persistent object is referenced in a command. The TRM is required to make sure that an object slot is available.  **CLEAR (0):** indicates that the TPM does not use transient-object slots when persistent objects are referenced |
| 31:3 | Reserved | shall be zero |

## TPMA\_CC (Command Code Attributes)

### Introduction

This structure defines the attributes of a command from a context management perspective. The fields of the structure indicate to the TPM Resource Manager (TRM) the number of resources required by a command and how the command affects the TPM’s resources.

This structure is only used in a list returned by the TPM in response to TPM2\_GetCapability(capability = TPM\_CAP\_COMMANDS).

For a command to the TPM, only the *commandIndex* field and *V* attribute are allowed to be non-zero.

### Structure Definition

Table 37 — Definition of (TPM\_CC) TPMA\_CC Bits <OUT>

| Bit | Name | Definition |
| --- | --- | --- |
| 15:0 | commandIndex | indicates the command being selected |
| 21:16 | Reserved | shall be zero |
| 22 | nv | **SET (1):** indicates that the command may write to NV  **CLEAR (0):** indicates that the command does not write to NV |
| 23 | extensive | **SET (1):** This command could flush any number of loaded contexts.  **CLEAR (0):** no additional changes other than indicated by the *flushed* attribute |
| 24 | flushed | **SET (1):** The context associated with any transient handle in the command will be flushed when this command completes.  **CLEAR (0):** No context is flushed as a side effect of this command. |
| 27:25 | cHandles | indicates the number of the handles in the handle area for this command |
| 28 | rHandle | **SET (1):** indicates the presence of the handle area in the response |
| 29 | V | **SET (1):** indicates that the command is vendor-specific  **CLEAR (0):** indicates that the command is defined in a version of this specification |
| 31:30 | Res | allocated for software; shall be zero |

### Field Descriptions

#### Bits[15:0] – *commandIndex*

This is the command index of the command in the set of commands. The two sets are defined by the *V* attribute. If *V* is zero, then the *commandIndex* shall be in the set of commands defined in a version of this specification. If *V* is one, then the meaning of *commandIndex* is as determined by the TPM vendor.

#### Bit[22] – *nv*

If this attribute is SET, then the TPM may perform an NV write as part of the command actions. This write is independent of any write that may occur as a result of dictionary attack protection. If this attribute is CLEAR, then the TPM shall not perform an NV write as part of the command actions.

#### Bit[23] – *extensive*

If this attribute is SET, then the TPM may flush many transient objects as a side effect of this command. In TPM 2.0 Part 3, a command that has this attribute is indicated by using a “{E}” decoration in the “Description” column of the *commandCode* parameter.

EXAMPLE See “TPM2\_Clear” in TPM 2.0 Part 3.

NOTE The “{E}” decoration may be combined with other decorations such as “{NV}” in which case the decoration would be “{NV E}.”

#### Bit[24] – *flushed*

If this attribute is SET, then the TPM will flush transient objects as a side effect of this command. Any transient objects listed in the handle area of the command will be flushed from TPM memory. Handles associated with persistent objects, sessions, PCR, or other fixed TPM resources are not flushed.

NOTE The TRM is expected to use this value to determine how many objects are loaded into transient TPM memory.

NOTE The “{F}” decoration may be combined with other decorations such as “{NV}” in which case the decoration would be “{NV F}.”

If this attribute is SET for a command, and the handle of the command is associated with a hierarchy (TPM\_RH\_PLATFORM, TPM\_RH\_OWNER, or TPM\_RH\_ENDORSEMENT), all loaded objects in the indicated hierarchy are flushed.

The TRM is expected to know the behaviour of TPM2\_ContextSave(), and sessions are flushed when context saved, but objects are not. The *flushed* attribute for that command shall be CLEAR.

In TPM 2.0 Part 3, a command that has this attribute is indicated by using a “{F}” decoration in the “Description” column of the *commandCode* parameter.

EXAMPLE See “TPM2\_SequenceComplete” in TPM 2.0 Part 3.”

#### Bits[27:25] – *cHandles*

This field indicates the number of handles in the handle area of the command. This number allows the TRM to enumerate the handles in the handle area and find the position of the authorizations (if any).

#### Bit[28] – *rHandle*

If this attribute is SET, then the response to this command has a handle area. This area will contain no more than one handle. This field is necessary to allow the TRM to locate the *parameterSize* field in the response, which is then used to locate the authorizations.

NOTE The TRM is expected to “virtualize” the handle value for any returned handle.

A TPM command is only allowed to have one handle in the response handle area.

#### Bit[29] – V

When this attribute is SET, it indicates that the command operation is defined by the TPM vendor. When CLEAR, it indicates that the command is defined by a version of this specification.

#### Bits[31:30] – Res

This field is reserved for system software. This field is required to be zero for a command to the TPM.

## TPMA\_MODES

This structure of this attribute is used to report that the TPM is designed for these modes. This structure may be read using TPM2\_GetCapability(*capability* = TPM\_CAP\_TPM\_PROPERTIES, *property* = TPM\_PT\_MODES).

NOTE: To determine the certification status of a TPM with the FIPS\_140\_2 attribute SET, consult the NIST Module Validation List at http://csrc.nist.gov/groups/STM/cmvp/validation.html.

Table 38 — Definition of (UINT32) TPMA\_MODES Bits <Out>

|  |  |  |
| --- | --- | --- |
| Bit | Name | Definition |
| 0 | FIPS\_140\_2 | **SET (1):** indicates that the TPM is designed to comply with all of the FIPS 140-2 requirements at Level 1 or higher. |
| 31:1 | Reserved | shall be zero |

## TPMA\_X509\_KEY\_USAGE

These attributes are as specified in clause 4.2.1.3. of RFC 5280 *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* For TPM2\_CertifyX509, when a caller provides a DER encoded Key Usage in *partialCertificate*, the TPM will validate that the key to be certified meets the requirements of Key Usage.

RFC 5280 describes these attributes in terms of how the public key in the certificate should be used. The TPM needs to check that the attributes of the key allow the private part of the key to be used for a purpose that is complimentary to the use of the public key. That is, if the public key should be used to verify signatures, the private key needs to be able to create the signatures (have *sign* SET).

This structure is defined to provide labels of the attributes for use by the TPM code that validates the attributes. This structure is input to the TPM as a DER encoded structure and not in the normal, TPM-canonical form.

This structure is only input to the TPM in a DER-encoded structure and is not present on the interface in canonical TPM format.

Table 39 — Definition of (UINT32) TPMA\_X509\_KEY\_USAGE Bits<>

|  |  |  |
| --- | --- | --- |
| Bit | Atrribute | Requirements |
| 22:0 | Reserved |  |
| 23 | decipherOnly | Attributes.Decrypt SET |
| 24 | encipherOnly | Attributes.Decrypt SET |
| 25 | cRLSign | Attributes.sign SET |
| 26 | keyCertSign | Attributes.sign SET |
| 27 | keyAgreement | Attributes.Decrypt SET |
| 28 | dataEncipherment | Attributes.Decrypt SET |
| 29 | keyEncipherment | asymmetric key with *decrypt* and *restricted* SET – key has the attributes of a parent key |
| 30 | nonrepudiation/contentCommitment | *fixedTPM* SET in Subject Key (*objectHandle*) |
| 31 | digitalSignature | *sign* SET in Subject Key (*objectHandle*) |

## TPMA\_ACT

This attribute is used to report the ACT state. This attribute may be read using TPM2\_GetCapability(*capability* = TPM\_CAP\_ACT, *property* = TPM\_RH\_ACT\_”x” where “x” is the ACT number (0-F)). The *signaled* value must be preserved across TPM Resume or if the TPM has not lost power. The *signaled* value may be preserved over a power cycle of a TPM.

NOTE: The ACT signaled value is reset to zero when the ACT is next accessed by TPM2\_ACT\_SetTimeout() with a non-zero *startTimeout*.

Table 40 — Definition of (UINT32) TPMA\_ACT Bits

| Bit | Name | Definition |
| --- | --- | --- |
| 0 | signaled | **SET (1):** The ACT has signaled  **CLEAR (0):** The ACT has not signaled |
| 1 | preserveSignaled | **SET (1):** The ACT signaled bit is preserved over a power cycle  **CLEAR (0):** The ACT signaled bit is not preserved over a power cycle |
| 31:2 | Reserved | shall be zero |

# Interface Types

## Introduction

Clause 8.11 contains definitions for interface types. An interface type is type checked when it is unmarshaled. These types are based on an underlying type that is indicated in the table title by the value in parentheses. When an interface type is used, the base type is unmarshaled and then checked to see if it has one of the allowed values.

## TPMI\_YES\_NO

This interface type is used in place of a Boolean type in order to eliminate ambiguity in the handling of a octet that conveys a single bit of information. This type only has two allowed values, YES (1) and NO (0).

NOTE This list is not used as input to the TPM.

Table 41 — Definition of (BYTE) TPMI\_YES\_NO Type

|  |  |
| --- | --- |
| Value | Description |
| NO | a value of 0 |
| YES | a value of 1 |
| #TPM\_RC\_VALUE |  |

## TPMI\_DH\_OBJECT

The TPMI\_DH\_OBJECT interface type is a handle that references a loaded object. The handles in this set are used to refer to either transient or persistent object. The range of these values would change according to the TPM implementation.

NOTE These interface types should not be used by system software to qualify the keys produced by the TPM. The value returned by the TPM shall be used to reference the object.

Table 42 — Definition of (TPM\_HANDLE) TPMI\_DH\_OBJECT Type

| Values | Comments |
| --- | --- |
| {TRANSIENT\_FIRST:TRANSIENT\_LAST} | allowed range for transient objects |
| {PERSISTENT\_FIRST:PERSISTENT\_LAST} | allowed range for persistent objects |
| +TPM\_RH\_NULL | the conditional value |
| #TPM\_RC\_VALUE |  |

## TPMI\_DH\_PARENT

The TPMI\_DH\_PARENT interface type is a handle that references an object that can be the parent of another object. The handles in this set may refer to either transient or persistent object or to Primary Seeds.

Table 43 — Definition of (TPM\_HANDLE) TPMI\_DH\_PARENT Type

| Values | Comments |
| --- | --- |
| {TRANSIENT\_FIRST:TRANSIENT\_LAST} | allowed range for transient objects |
| {PERSISTENT\_FIRST:PERSISTENT\_LAST} | allowed range for persistent objects |
| TPM\_RH\_OWNER | Storage hierarchy |
| TPM\_RH\_PLATFORM | Platform hierarchy |
| TPM\_RH\_ENDORSEMENT | Endorsement hierarchy |
| +TPM\_RH\_NULL | no hierarchy |
| #TPM\_RC\_VALUE |  |

## TPMI\_DH\_PERSISTENT

The TPMI\_DH\_PERSISTENT interface type is a handle that references a location for a transient object. This type is used in TPM2\_EvictControl() to indicate the handle to be assigned to the persistent object.

Table 44 — Definition of (TPM\_HANDLE) TPMI\_DH\_PERSISTENT Type

| Values | Comments |
| --- | --- |
| {PERSISTENT\_FIRST:PERSISTENT\_LAST} | allowed range for persistent objects |
| #TPM\_RC\_VALUE |  |

## TPMI\_DH\_ENTITY

The TPMI\_DH\_ENTITY interface type is TPM-defined values that are used to indicate that the handle refers to an *authValue*. The range of these values would change according to the TPM implementation.

Table 45 — Definition of (TPM\_HANDLE) TPMI\_DH\_ENTITY Type <IN>

| Values | Comments |
| --- | --- |
| TPM\_RH\_OWNER |  |
| TPM\_RH\_ENDORSEMENT |  |
| TPM\_RH\_PLATFORM |  |
| TPM\_RH\_LOCKOUT |  |
| {TRANSIENT\_FIRST : TRANSIENT\_LAST} | range of object handles |
| {PERSISTENT\_FIRST : PERSISTENT\_LAST} |  |
| {NV\_INDEX\_FIRST : NV\_INDEX\_LAST} |  |
| {PCR\_FIRST : PCR\_LAST} |  |
| {TPM\_RH\_AUTH\_00 : TPM\_RH\_AUTH\_FF} | range of vendor-specific authorization values |
| +TPM\_RH\_NULL | conditional value |
| #TPM\_RC\_VALUE |  |

## TPMI\_DH\_PCR

This interface type consists of the handles that may be used as PCR references. The upper end of this range of values would change according to the TPM implementation.

NOTE 1 Typically, the 0th PCR will have a handle value of zero.

NOTE 2 The handle range for PCR is defined to be the same as the handle range for PCR in previous versions of TPM specifications.

Table 46 — Definition of (TPM\_HANDLE) TPMI\_DH\_PCR Type <IN>

| Values | Comments |
| --- | --- |
| {PCR\_FIRST:PCR\_LAST} |  |
| +TPM\_RH\_NULL | conditional value |
| #TPM\_RC\_VALUE |  |

## TPMI\_SH\_AUTH\_SESSION

The TPMI\_SH\_AUTH\_SESSION interface type is TPM-defined values that are used to indicate that the handle refers to an authorization session.

Table 47 — Definition of (TPM\_HANDLE) TPMI\_SH\_AUTH\_SESSION Type <IN/OUT>

| Values | Comments |
| --- | --- |
| {HMAC\_SESSION\_FIRST : HMAC\_SESSION\_LAST} | range of HMAC authorization session handles |
| {POLICY\_SESSION\_FIRST: POLICY\_SESSION\_LAST} | range of policy authorization session handles |
| +TPM\_RS\_PW | a password authorization |
| #TPM\_RC\_VALUE | error returned if the handle is out of range |

## TPMI\_SH\_HMAC

This interface type is used for an authorization handle when the authorization session uses an HMAC.

Table 48 — Definition of (TPM\_HANDLE) TPMI\_SH\_HMAC Type <IN/OUT>

| Values | Comments |
| --- | --- |
| {HMAC\_SESSION\_FIRST: HMAC\_SESSION\_LAST} | range of HMAC authorization session handles |
| #TPM\_RC\_VALUE | error returned if the handle is out of range |

## TPMI\_SH\_POLICY

This interface type is used for a policy handle when it appears in a policy command.

Table 49 — Definition of (TPM\_HANDLE) TPMI\_SH\_POLICY Type <IN/OUT>

| Values | Comments |
| --- | --- |
| {POLICY\_SESSION\_FIRST: POLICY\_SESSION\_LAST} | range of policy authorization session handles |
| #TPM\_RC\_VALUE | error returned if the handle is out of range |

## TPMI\_DH\_CONTEXT

This type defines the handle values that may be used in TPM2\_ContextSave() or TPM2\_Flush().

Table 50 — Definition of (TPM\_HANDLE) TPMI\_DH\_CONTEXT Type

| Values | Comments |
| --- | --- |
| {HMAC\_SESSION\_FIRST : HMAC\_SESSION\_LAST} |  |
| {POLICY\_SESSION\_FIRST:POLICY\_SESSION\_LAST} |  |
| {TRANSIENT\_FIRST:TRANSIENT\_LAST} |  |
| #TPM\_RC\_VALUE |  |

## TPMI\_DH\_SAVED

This type defines the handle values that may be used in TPM2\_ContextSave() or TPM2\_FlushContext().

Table 51 — Definition of (TPM\_HANDLE) TPMI\_DH\_SAVED Type

| Values | Comments |
| --- | --- |
| {HMAC\_SESSION\_FIRST : HMAC\_SESSION\_LAST} | an HMAC session context |
| {POLICY\_SESSION\_FIRST:POLICY\_SESSION\_LAST} | a policy session context |
| 0x80000000 | an ordinary transient object |
| 0x80000001 | a sequence object |
| 0x80000002 | a transient object with the *stClear* attribute SET |
| #TPM\_RC\_VALUE |  |

## TPMI\_RH\_HIERARCHY

The TPMI\_RH\_HIERARCHY interface type is used as the type of a handle in a command when the handle is required to be one of the hierarchy selectors.

Table 52 — Definition of (TPM\_HANDLE) TPMI\_RH\_HIERARCHY Type

| Values | Comments |
| --- | --- |
| TPM\_RH\_OWNER | Storage hierarchy |
| TPM\_RH\_PLATFORM | Platform hierarchy |
| TPM\_RH\_ENDORSEMENT | Endorsement hierarchy |
| +TPM\_RH\_NULL | no hierarchy |
| #TPM\_RC\_VALUE | response code returned when the unmarshaling of this type fails |

## TPMI\_RH\_ENABLES

The TPMI\_RH\_ENABLES interface type is used as the type of a handle in a command when the handle is required to be one of the hierarchy or NV enables.

Table 53 — Definition of (TPM\_HANDLE) TPMI\_RH\_ENABLES Type

| Values | Comments |
| --- | --- |
| TPM\_RH\_OWNER | Storage hierarchy |
| TPM\_RH\_PLATFORM | Platform hierarchy |
| TPM\_RH\_ENDORSEMENT | Endorsement hierarchy |
| TPM\_RH\_PLATFORM\_NV | Platform NV |
| +TPM\_RH\_NULL | no hierarchy |
| #TPM\_RC\_VALUE | response code returned when the unmarshaling of this type fails |

## TPMI\_RH\_HIERARCHY\_AUTH

This interface type is used as the type of a handle in a command when the handle is required to be one of the hierarchy selectors or the Lockout Authorization.

Table 54 — Definition of (TPM\_HANDLE) TPMI\_RH\_HIERARCHY\_AUTH Type <IN>

| Values | Comments |
| --- | --- |
| TPM\_RH\_OWNER | Storage hierarchy |
| TPM\_RH\_PLATFORM | Platform hierarchy |
| TPM\_RH\_ENDORSEMENT | Endorsement hierarchy |
| TPM\_RH\_LOCKOUT | Lockout Authorization |
| #TPM\_RC\_VALUE | response code returned when the unmarshaling of this type fails |

## TPMI\_RH\_HIERARCHY\_POLICY

This interface type is used as the type of a handle in a command when the handle is required to be one of the hierarchy selectors, the Lockout Authorization, or an ACT. This type is used in TPM2\_SetPrimaryPolicy().

Table 55 — Definition of (TPM\_HANDLE) TPMI\_RH\_HIERARCHY\_POLICY Type <IN>

| Values | Comments |
| --- | --- |
| TPM\_RH\_OWNER | Storage hierarchy |
| TPM\_RH\_PLATFORM | Platform hierarchy |
| TPM\_RH\_ENDORSEMENT | Endorsement hierarchy |
| TPM\_RH\_LOCKOUT | Lockout Authorization |
| {TPM\_RH\_ACT\_0:TPM\_RH\_ACT\_F} | Authenticated Countdown Timer |
| #TPM\_RC\_VALUE | response code returned when the unmarshaling of this type fails |

## TPMI\_RH\_PLATFORM

The TPMI\_RH\_PLATFORM interface type is used as the type of a handle in a command when the only allowed handle is TPM\_RH\_PLATFORM indicating that Platform Authorization is required.

Table 56 — Definition of (TPM\_HANDLE) TPMI\_RH\_PLATFORM Type <IN>

| Values | Comments |
| --- | --- |
| TPM\_RH\_PLATFORM | Platform hierarchy |
| #TPM\_RC\_VALUE | response code returned when the unmarshaling of this type fails |

## TPMI\_RH\_OWNER

This interface type is used as the type of a handle in a command when the only allowed handle is TPM\_RH\_OWNER indicating that Owner Authorization is required.

Table 57 — Definition of (TPM\_HANDLE) TPMI\_RH\_OWNER Type <IN>

| Values | Comments |
| --- | --- |
| TPM\_RH\_OWNER | Owner hierarchy |
| +TPM\_RH\_NULL | may allow the null handle |
| #TPM\_RC\_VALUE | response code returned when the unmarshaling of this type fails |

## TPMI\_RH\_ENDORSEMENT

This interface type is used as the type of a handle in a command when the only allowed handle is TPM\_RH\_ENDORSEMENT indicating that Endorsement Authorizationis required.

Table 58 — Definition of (TPM\_HANDLE) TPMI\_RH\_ENDORSEMENT Type <IN>

| Values | Comments |
| --- | --- |
| TPM\_RH\_ENDORSEMENT | Endorsement hierarchy |
| +TPM\_RH\_NULL | may allow the null handle |
| #TPM\_RC\_VALUE | response code returned when the unmarshaling of this type fails |

## TPMI\_RH\_PROVISION

The TPMI\_RH\_PROVISION interface type is used as the type of the handle in a command when the only allowed handles are either TPM\_RH\_OWNER or TPM\_RH\_PLATFORM indicating that either Platform Authorization or Owner Authorization are allowed.

In most cases, either Platform Authorization or Owner Authorization may be used to authorize the commands used for management of the resources of the TPM and this interface type will be used.

Table 59 — Definition of (TPM\_HANDLE) TPMI\_RH\_PROVISION Type <IN>

| Value | Comments |
| --- | --- |
| TPM\_RH\_OWNER | handle for Owner Authorization |
| TPM\_RH\_PLATFORM | handle for Platform Authorization |
| #TPM\_RC\_VALUE | response code returned when the unmarshaling of this type fails |

## TPMI\_RH\_CLEAR

The TPMI\_RH\_CLEAR interface type is used as the type of the handle in a command when the only allowed handles are either TPM\_RH\_LOCKOUT or TPM\_RH\_PLATFORM indicating that either Platform Authorization or Lockout Authorization are allowed.

This interface type is normally used for performing or controlling TPM2\_Clear().

Table 60 — Definition of (TPM\_HANDLE) TPMI\_RH\_CLEAR Type <IN>

| Value | Comments |
| --- | --- |
| TPM\_RH\_LOCKOUT | handle for Lockout Authorization |
| TPM\_RH\_PLATFORM | handle for Platform Authorization |
| #TPM\_RC\_VALUE | response code returned when the unmarshaling of this type fails |

## TPMI\_RH\_NV\_AUTH

This interface type is used to identify the source of the authorization for access to an NV location. The handle value of a TPMI\_RH\_NV\_AUTH shall indicate that the authorization value is either Platform Authorization, Owner Authorization, or the *authValue.* This type is used in the commands that access an NV Index (commands of the form TPM2\_NV\_xxx) other than TPM2\_NV\_DefineSpace() and TPM2\_NV\_UndefineSpace().

Table 61 — Definition of (TPM\_HANDLE) TPMI\_RH\_NV\_AUTH Type <IN>

| Value | Comments |
| --- | --- |
| TPM\_RH\_PLATFORM | Platform Authorization is allowed |
| TPM\_RH\_OWNER | Owner Authorization is allowed |
| {NV\_INDEX\_FIRST:NV\_INDEX\_LAST} | range for NV locations |
| #TPM\_RC\_VALUE | response code returned when unmarshaling of this type fails |

## TPMI\_RH\_LOCKOUT

The TPMI\_RH\_LOCKOUT interface type is used as the type of a handle in a command when the only allowed handle is TPM\_RH\_LOCKOUT indicating that Lockout Authorization is required.

Table 62 — Definition of (TPM\_HANDLE) TPMI\_RH\_LOCKOUT Type <IN>

| Value | Comments |
| --- | --- |
| TPM\_RH\_LOCKOUT | handle for Lockout Authorization |
| #TPM\_RC\_VALUE | response code returned when the unmarshaling of this type fails |

## TPMI\_RH\_NV\_INDEX

This interface type is used to identify an NV location. This type is used in the NV commands.

Table 63 — Definition of (TPM\_HANDLE) TPMI\_RH\_NV\_INDEX Type <IN/OUT>

| Value | Comments |
| --- | --- |
| {NV\_INDEX\_FIRST:NV\_INDEX\_LAST} | Range of NV Indexes |
| #TPM\_RC\_VALUE | error returned if the handle is out of range |

## TPMI\_RH\_AC

This interface type is used to identify an attached component. This type is used in the AC commands.

Table 64 — Definition of (TPM\_HANDLE) TPMI\_RH\_AC Type <IN>

| Value | Comments |
| --- | --- |
| {AC\_FIRST:AC\_LAST} | Range of AC handles |
| #TPM\_RC\_VALUE | error returned if the handle is out of range |

## TPMI\_RH\_ACT

This interface type is used to identify the ACT instance used in TPM2\_ACT\_SetTimeout().

Table 65 — Definition of (TPM\_HANDLE) TPMI\_RH\_ACT Type

| Value | Comments |
| --- | --- |
| {TPM\_RH\_ACT\_0:TPM\_RH\_ACT\_F} | handles for the Authenticated Countdown Timers |
| #TPM\_RC\_VALUE | response code returned when the unmarshaling of this type fails |

## TPMI\_ALG\_HASH

A TPMI\_ALG\_HASH is an interface type of all the hash algorithms implemented on a specific TPM. The selector in Table 66 indicates all of the hash algorithms that have an algorithm ID assigned by the TCG and does not indicate the algorithms that will be accepted by a TPM.

NOTE When implemented, each of the algorithm entries is delimited by #ifdef and #endif so that, if the algorithm is not implemented in a specific TPM, that algorithm is not included in the interface type.

Table 66 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_HASH Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.H | all hash algorithms defined by the TCG |
| +TPM\_ALG\_NULL |  |
| #TPM\_RC\_HASH |  |

## TPMI\_ALG\_ASYM (Asymmetric Algorithms)

A TPMI\_ALG\_ASYM is an interface type of all the asymmetric algorithms implemented on a specific TPM. Table 67 lists each of the asymmetric algorithms that have an algorithm ID assigned by the TCG.

Table 67 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_ASYM Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.AO | all asymmetric object types |
| +TPM\_ALG\_NULL |  |
| #TPM\_RC\_ASYMMETRIC |  |

## TPMI\_ALG\_SYM (Symmetric Algorithms)

A TPMI\_ALG\_SYM is an interface type of all the symmetric algorithms that have an algorithm ID assigned by the TCG and are implemented on the TPM.

NOTE The validation code produced by an example script will produce a CASE statement with a case for each of the values in the “Values” column. The case for a value is delimited by a #ifdef/#endif pair so that if the algorithm is not implemented on the TPM, then the case for the algorithm is not generated, and use of the algorithm will cause a TPM error (TPM\_RC\_SYMMETRIC).

Table 68 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_SYM Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.S | all symmetric block ciphers |
| TPM\_ALG\_XOR | required |
| +TPM\_ALG\_NULL | required to be present in all versions of this table |
| #TPM\_RC\_SYMMETRIC |  |

## TPMI\_ALG\_SYM\_OBJECT

A TPMI\_ALG\_SYM\_OBJECT is an interface type of all the TCG-defined symmetric algorithms that may be used as companion symmetric encryption algorithm for an asymmetric object. All algorithms in this list shall be block ciphers usable in Cipher Feedback (CFB).

NOTE TPM\_ALG\_XOR is not allowed in this list.

Table 69 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_SYM\_OBJECT Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.S | all symmetric block ciphers |
| +TPM\_ALG\_NULL | required to be present in all versions of this table |
| #TPM\_RC\_SYMMETRIC |  |

## TPMI\_ALG\_SYM\_MODE

A TPMI\_ALG\_SYM\_MODE is an interface type of all the TCG-defined block-cipher modes of operation.

Table 70 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_SYM\_MODE Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.SE | all symmetric block cipher encryption/decryption modes |
| TPM\_ALG\_!ALG.SX | all symmetric block cipher MAC modes |
| +TPM\_ALG\_NULL |  |
| #TPM\_RC\_MODE |  |

## TPMI\_ALG\_KDF (Key and Mask Generation Functions)

A TPMI\_ALG\_KDF is an interface type of all the key derivation functions implemented on a specific TPM.

Table 71 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_KDF Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.HM | all defined hash-based key and mask generation functions |
| +TPM\_ALG\_NULL |  |
| #TPM\_RC\_KDF |  |

## TPMI\_ALG\_SIG\_SCHEME

This is the definition of the interface type for any signature scheme.

Table 72 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_SIG\_SCHEME Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.ax | all asymmetric signing schemes including anonymous schemes |
| TPM\_ALG\_HMAC | present on all TPM |
| +TPM\_ALG\_NULL |  |
| #TPM\_RC\_SCHEME | response code when a signature scheme is not correct |

## TPMI\_ECC\_KEY\_EXCHANGE

This is the definition of the interface type for an ECC key exchange scheme.

NOTE Because of the “{ECC}” in the table title, the only values in this table will be those that are dependent on ECC being implemented, even if they otherwise have the correct type attributes.

Table 73 — Definition of (TPM\_ALG\_ID){ECC} TPMI\_ECC\_KEY\_EXCHANGE Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.AM | any ECC key exchange method |
| TPM\_ALG\_SM2 | SM2 is typed as signing but may be used as a key-exchange protocol |
| +TPM\_ALG\_NULL |  |
| #TPM\_RC\_SCHEME | response code when a key exchange scheme is not correct |

## TPMI\_ST\_COMMAND\_TAG

This interface type is used for the command tags.

The response code for a bad command tag has the same value as the TPM 1.2 response code (TPM\_BAD\_TAG). This value is used in case the software is not compatible with this specification and an unexpected response code might have unexpected side effects.

Table 74 — Definition of (TPM\_ST) TPMI\_ST\_COMMAND\_TAG Type

| Values | Comments |
| --- | --- |
| TPM\_ST\_NO\_SESSIONS |  |
| TPM\_ST\_SESSIONS |  |
| #TPM\_RC\_BAD\_TAG |  |

## TPMI\_ALG\_MAC\_SCHEME

A TPMI\_ALG\_MAC\_SCHEME is an interface type of all the TCG-defined symmetric algorithms that may be used as companion symmetric! signing algorithm.

Table 75 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_MAC\_SCHEME Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.SX | all symmetric block cipher MAC algorithms |
| TPM\_ALG\_!ALG.H | all hash algorithms defined by the TCG |
| +TPM\_ALG\_NULL | required to be present in all versions of this table |
| #TPM\_RC\_SYMMETRIC |  |

## TPMI\_ALG\_CIPHER\_MODE

A TPMI\_ALG\_CIPHER\_MODE is an interface type of all the symmetric block cipher, encryption/decryption modes that are listed in the TCG algorithm registry.

Table 76 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_CIPHER\_MODE Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.SE | all symmetric block cipher algorithms |
| +TPM\_ALG\_NULL | required to be present in all versions of this table |
| #TPM\_RC\_MODE |  |

# Structure Definitions

## TPMS\_EMPTY

This structure is used as a placeholder. In some cases, a union will have a selector value with no data to unmarshal when that type is selected. Rather than leave the entry empty, TPMS\_EMPTY may be selected.

NOTE The tool chain will special case this structure and create the marshaling and unmarshaling code for this structure but not create a type definition. The unmarshaling code for this structure will return TPM\_RC\_SUCCESS and the marshaling code will return 0.

Table 77 — Definition of TPMS\_EMPTY Structure <IN/OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
|  |  | a structure with no member |

## TPMS\_ALGORITHM\_DESCRIPTION

This structure is a return value for a TPM2\_GetCapability() that reads the installed algorithms.

Table 78 — Definition of TPMS\_ALGORITHM\_DESCRIPTION Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| alg | TPM\_ALG\_ID | an algorithm |
| attributes | TPMA\_ALGORITHM | the attributes of the algorithm |

## Hash/Digest Structures

### TPMU\_HA (Hash)

A TPMU\_HA is a union of all the hash algorithms implemented on a TPM.

NOTE 1 The !ALG.H and !ALG.H values represent all algorithms defined in the TCG registry as being type “H”.

NOTE 2 If processed by an automated tool, each entry of the table should be qualified (with #ifdef/#endif) so that if the hash algorithm is not implemented on the TPM, the parameter associated with that hash is not present. This will keep the union from being larger than the largest digest of a hash implemented on that TPM.

Table 79 — Definition of TPMU\_HA Union <IN/OUT >

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| !ALG.H [!ALG.H\_DIGEST\_SIZE] | BYTE | TPM\_ALG\_!ALG.H | all hashes |
| null |  | TPM\_ALG\_NULL |  |

### TPMT\_HA

Table 80 shows the basic hash-agile structure used in this specification. To handle hash agility, this structure uses the *hashAlg* parameter to indicate the algorithm used to compute the digest and, by implication, the size of the digest.

When transmitted, only the number of octets indicated by *hashAlg* is sent.

NOTE In the reference code, when a TPMT\_HA is allocated, the digest field is large enough to support the largest hash algorithm in the TPMU\_HA union.

Table 80 — Definition of TPMT\_HA Structure <IN/OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| hashAlg | +TPMI\_ALG\_HASH | selector of the hash contained in the *digest* that implies the size of the *digest*  NOTE The leading “+” on the type indicates that this structure should pass an indication to the unmarshaling function for TPMI\_ALG\_HASH so that TPM\_ALG\_NULL will be allowed if a use of a TPMT\_HA allows TPM\_ALG\_NULL. |
| [hashAlg] digest | TPMU\_HA | the digest data |

## Sized Buffers

### Introduction

The “TPM2B\_” prefix is used for a structure that has a size field followed by a data buffer with the indicated number of octets. The *size* field is 16 bits.

When the type of the second parameter in a TPM2B\_ structure is BYTE, the TPM shall unmarshal the indicated number of octets, which may be zero.

When the type of the second parameter in the TPM2B\_ structure is not BYTE, the value of the *size* field shall either be zero indicating that no structure is to be unmarshaled; or it shall be identical to the number of octets unmarshaled for the second parameter.

NOTE 1 If the TPM2B\_ defines a structure and not an array of octets, then the structure is self-describing and the TPM will be able to determine how many octets are in the structure when it is unmarshaled. If that number of octets is not equal to the size parameter, then it is an error.

NOTE 2 The reason that a structure may be put into a TPM2B\_ is that the parts of the structure may be handled as separate opaque blocks by the application/system software. Rather than require that all of the structures in a command or response be marshaled or unmarshaled sequentially, the size field allows the structure to be manipulated as an opaque block. Placing a structure in a TPM2B\_ also makes it possible to use parameter encryption on the structure.

If a TPM2B\_ is encrypted, the TPM will encrypt/decrypt the data field of the TPM2B\_ but not the *size* parameter. The TPM will encrypt/decrypt the number of octets indicated by the *size* field.

NOTE 3 In the reference implementation, a TPM2B type is defined that is a 16-bit size field followed by a single byte of data. The TPM2B\_ is then defined as a union that contains a TPM2B (union member ‘b’) and the structure in the definition table (union member ‘t’). This union is used for internally generated structures so that there is a way to define a structure of the correct size (forced by the ‘t’ member) while giving a way to pass the structure generically as a ‘b’. Most function calls use the 't' member so that the compiler will generate a warning if there is a type error (a TPM2B\_ of the wrong type). Having the type checked helps avoid many issues with buffer overflow caused by a too small buffer being passed to a function.

### TPM2B\_DIGEST

This structure is used for a sized buffer that cannot be larger than the largest digest produced by any hash algorithm implemented on the TPM.

As with all sized buffers, the size is checked to see if it is within the prescribed range. If not, the response code is TPM\_RC\_SIZE.

NOTE For any structure, like the one below, that contains an implied size check, it is implied that TPM\_RC\_SIZE is a possible response code and the response code will not be listed in the table.

Table 81 — Definition of TPM2B\_DIGEST Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size in octets of the *buffer* field; may be 0 |
| buffer[size]{:sizeof(TPMU\_HA)} | BYTE | the buffer area that can be no larger than a digest |

### TPM2B\_DATA

This structure is used for a data buffer that is required to be no larger than the size of the Name of an object.

Table 82 — Definition of TPM2B\_DATA Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size in octets of the *buffer* field; may be 0 |
| buffer[size]{:sizeof(TPMT\_HA)} | BYTE |  |

### TPM2B\_NONCE

Table 83 — Definition of Types for TPM2B\_NONCE

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| TPM2B\_DIGEST | TPM2B\_NONCE | size limited to the same as the digest structure |

### TPM2B\_AUTH

This structure is used for an authorization value and limits an *authValue* to being no larger than the largest digest produced by a TPM. In order to ensure consistency within an object, the *authValue* may be no larger than the size of the digest produced by the object’s *nameAlg*. This ensures that any TPM that can load the object will be able to handle the *authValue* of the object.

Table 84 — Definition of Types for TPM2B\_AUTH

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| TPM2B\_DIGEST | TPM2B\_AUTH | size limited to the same as the digest structure |

### TPM2B\_OPERAND

This type is a sized buffer that can hold an operand for a comparison with an NV Index location. The maximum size of the operand is implementation dependent but a TPM is required to support an operand size that is at least as big as the digest produced by any of the hash algorithms implemented on the TPM.

Table 85 — Definition of Types for TPM2B\_OPERAND

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| TPM2B\_DIGEST | TPM2B\_OPERAND | size limited to the same as the digest structure |

### TPM2B\_EVENT

This type is a sized buffer that can hold event data.

Table 86 — Definition of TPM2B\_EVENT Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size of the operand *buffer* |
| buffer [size] {:1024} | BYTE | the operand |

### TPM2B\_MAX\_BUFFER

This type is a sized buffer that can hold a maximally sized buffer for commands that use a large data buffer such as TPM2\_Hash(), TPM2\_SequenceUpdate(), or TPM2\_FieldUpgradeData().

NOTE The above list is not comprehensive and other commands may use this buffer type.

MAX\_DIGEST\_BUFFER is TPM-dependent but is required to be at least 1,024.

Table 87 — Definition of TPM2B\_MAX\_BUFFER Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size of the buffer |
| buffer [size] {:MAX\_DIGEST\_BUFFER} | BYTE | the operand |

### TPM2B\_MAX\_NV\_BUFFER

This type is a sized buffer that can hold a maximally sized buffer for NV data commands such as TPM2\_NV\_Read(), TPM2\_NV\_Write(), and TPM2\_NV\_Certify().

Table 88 — Definition of TPM2B\_MAX\_NV\_BUFFER Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size of the buffer |
| buffer [size] {:MAX\_NV\_BUFFER\_SIZE} | BYTE | the operand  NOTE MAX\_NV\_BUFFER\_SIZE is TPM-dependent |

### TPM2B\_TIMEOUT

This TPM-dependent structure is used to provide the timeout value for an authorization. The *size* shall be 8 or less.

Table 89 — Definition of TPM2B\_TIMEOUT Structure

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| size | UINT16 | size of the timeout value |
| buffer[size]{:sizeof(UINT64)} | BYTE | the timeout value |

NOTE In the reference implementation the MSb is used as a flag to indicate whether a ticket expires on TPM Reset or TPM Restart.

### TPM2B\_IV

This structure is used for passing an initial value for a symmetric block cipher to or from the TPM. The size is set to be the largest block size of any implemented symmetric cipher implemented on the TPM.

Table 90 — Definition of TPM2B\_IV Structure <IN/OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size of the IV value  This value is fixed for a TPM implementation. |
| buffer[size]{:MAX\_SYM\_BLOCK\_SIZE} | BYTE | the IV value |

## Names

### Introduction

The Name of an entity is used in place of the handle in authorization computations. The substitution occurs in *cpHash* and *policyHash* computations.

For an entity that is defined by a public area (objects and NV Indexes), the Name is the hash of the public structure that defines the entity. The hash is done using the *nameAlg* of the entity.

NOTE For an object, a TPMT\_PUBLIC defines the entity. For an NV Index, a TPMS\_NV\_PUBLIC defines the entity.

For entities not defined by a public area, the Name is the handle that is used to refer to the entity.

### TPMU\_NAME

Table 91 — Definition of TPMU\_NAME Union <>

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| digest | TPMT\_HA |  | when the Name is a digest |
| handle | TPM\_HANDLE |  | when the Name is a handle |

### TPM2B\_NAME

This buffer holds a Name for any entity type.

The type of Name in the structure is determined by context and the *size* parameter. If *size* is four, then the Name is a handle. If *size* is zero, then no Name is present. Otherwise, the size shall be the size of a TPM\_ALG\_ID plus the size of the digest produced by the indicated hash algorithm.

Table 92 — Definition of TPM2B\_NAME Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size of the Name structure |
| name[size]{:sizeof(TPMU\_NAME)} | BYTE | the Name structure |

## PCR Structures

### TPMS\_PCR\_SELECT

This structure provides a standard method of specifying a list of PCR.

PCR numbering starts at zero.

*pcrSelect* is an array of octets. The octet containing the bit corresponding to a specific PCR is found by dividing the PCR number by 8.

EXAMPLE 1 The bit in *pcrSelect* corresponding to PCR 19 is in *pcrSelect* [2] (19/8 = 2).

The least significant bit in a octet is bit number 0. The bit in the octet associated with a PCR is the remainder after division by 8.

EXAMPLE 2 The bit in *pcrSelect* [2] corresponding to PCR 19 is bit 3 (19 mod 8). If *sizeofSelect* is 3, then the *pcrSelect* array that would specify PCR 19 and no other PCR is 00 00 0816.

Each bit in *pcrSelect* indicates whether the corresponding PCR is selected (1) or not (0). If the *pcrSelect* is all zero bits, then no PCR is selected.

*sizeofSelect* indicates the number of octets in *pcrSelect*. The allowable value for *sizeofSelect* is determined by the number of PCR required by the applicable platform-specific specification and the number of PCR implemented in the TPM. The minimum value for *sizeofSelect* is:

PCR\_SELECT\_MIN ≔ (PLATFORM\_PCR + 7) / 8 1

where

PLATFORM\_PCR the number of PCR required by the platform-specific specification

The maximum value for sizeofSelect is:

PCR\_SELECT\_MAX ≔ (IMPLEMENTATION\_PCR + 7) / 8 1

where

IMPLEMENTATION\_PCR the number of PCR implemented on the TPM

If the TPM implements more PCR than there are bits in *pcrSelect*, the additional PCR are not selected.

EXAMPLE 3 If the applicable platform-specific specification requires that the TPM have a minimum of 24 PCR but the TPM implements 32, then a PCR select of 3 octets would imply that PCR 24-31 are not selected.

Table 93 — Definition of TPMS\_PCR\_SELECT Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| sizeofSelect {PCR\_SELECT\_MIN:} | UINT8 | the size in octets of the *pcrSelect* array |
| pcrSelect [sizeofSelect] {:PCR\_SELECT\_MAX} | BYTE | the bit map of selected PCR |
| #TPM\_RC\_VALUE |  |  |

### TPMS\_PCR\_SELECTION

Table 94 — Definition of TPMS\_PCR\_SELECTION Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| hash | TPMI\_ALG\_HASH | the hash algorithm associated with the selection |
| sizeofSelect {PCR\_SELECT\_MIN:} | UINT8 | the size in octets of the *pcrSelect* array |
| pcrSelect [sizeofSelect] {:PCR\_SELECT\_MAX} | BYTE | the bit map of selected PCR |
| #TPM\_RC\_VALUE |  |  |

## Tickets

### Introduction

Tickets are evidence that the TPM has previously processed some information. A ticket is an HMAC over the data using a secret key known only to the TPM. A ticket is a way to expand the state memory of the TPM. A ticket is only usable by the TPM that produced it.

The formulations for tickets shown in 10.7 are to be used by a TPM that is compliant with this specification.

The method of creating the ticket data is:

**HMAC**contexAlg(proof, (ticketType || param { || param {…})) 1

where

**HMAC**contexAlg() an HMAC using the hash used for context integrity

proof a TPM secret value (depends on hierarchy)

ticketType a value to differentiate the tickets

param one or more values that were checked by the TPM

The proof value used for each hierarchy is shown in Table 95.

Table 95 — Values for *proof* Used in Tickets

|  |  |  |
| --- | --- | --- |
| Hierarchy | proof | Description |
| Null | nullProof | a value that changes with every TPM Reset |
| Platform | phProof | a value that changes with each change of the PPS |
| Owner | shProof | a value that changes with each change of the SPS |
| Endorsement | ehProof | a value that changes with each change of either the EPS or SPS |

The format for a ticket is shown in Table 96. This is a template for the tickets shown in the remainder of this clause 10.7.

Table 96 — General Format of a Ticket

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| tag | TPM\_ST | structure tag indicating the type of the ticket |
| hierarchy | TPMI\_RH\_HIERARCHY+ | the hierarchy of the proof value |
| digest | TPM2B\_DIGEST | the HMAC over the ticket-specific data |

### A NULL Ticket

When a command requires a ticket and no ticket is available, the caller is required to provide a structure with a ticket *tag* that is correct for the context. The *hierarchy* shall be set to TPM\_RH\_NULL, and *digest* shall be the Empty Buffer (a buffer with a size field of zero). This construct is the NULL Ticket. When a response indicates that a ticket is returned, the TPM may return a NULL Ticket.

NOTE Because each use of a ticket requires that the structure tag for the ticket be appropriate for the use, there is no single representation of a NULL Ticket that will work in all circumstances. Minimally, a NULL ticket will have a structure type that is appropriate for the context.

### TPMT\_TK\_CREATION

This ticket is produced by TPM2\_Create() or TPM2\_CreatePrimary(). It is used to bind the creation data to the object to which it applies. The ticket is computed by

**HMAC**contextAlg(proof, (TPM\_ST\_CREATION || name || **H**nameAlg(TPMS\_CREATION\_DATA))) 1

where

**HMAC**contextAlg() an HMAC using the context integrity hash algorithm

proof a TPM secret value associated with the hierarchy associated with name

TPM\_ST\_CREATION a value used to ensure that the ticket is properly used

name the Name of the object to which the creation data is to be associated

**H**nameAlg() hash using the nameAlg of the created object

TPMS\_CREATION\_DATA the creation data structure associated with name

Table 97 — Definition of TPMT\_TK\_CREATION Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| tag {TPM\_ST\_CREATION} | TPM\_ST | ticket structure tag |
| #TPM\_RC\_TAG |  | error returned when *tag* is not TPM\_ST\_CREATION |
| hierarchy | TPMI\_RH\_HIERARCHY+ | the hierarchy containing *name* |
| digest | TPM2B\_DIGEST | This shall be the HMAC produced using a proof value of *hierarchy*. |

EXAMPLE A NULL Creation Ticket is the tuple <TPM\_ST\_CREATION, TPM\_RH\_NULL, 0x0000>.

### TPMT\_TK\_VERIFIED

This ticket is produced by TPM2\_VerifySignature(). This formulation is used for multiple ticket uses. The ticket provides evidence that the TPM has validated that a digest was signed by a key with the Name of keyName. The ticket is computed by

**HMAC**contextAlg(proof, (TPM\_ST\_VERIFIED || digest || keyName)) 1

where

**HMAC**contextAlg() an HMAC using the context integrity hash

proof a TPM secret value associated with the hierarchy associated with keyName

TPM\_ST\_VERIFIED a value used to ensure that the ticket is properly used

digest the signed digest

keyName Name of the key that signed digest

Table 98 — Definition of TPMT\_TK\_VERIFIED Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| tag {TPM\_ST\_VERIFIED} | TPM\_ST | ticket structure tag |
| #TPM\_RC\_TAG |  | error returned when *tag* is not TPM\_ST\_VERIFIED |
| hierarchy | TPMI\_RH\_HIERARCHY+ | the hierarchy containing *keyName* |
| digest | TPM2B\_DIGEST | This shall be the HMAC produced using a proof value of *hierarchy*. |

EXAMPLE A NULL Verified Ticket is the tuple <TPM\_ST\_VERIFIED, TPM\_RH\_NULL, 0x0000>.

### TPMT\_TK\_AUTH

This ticket is produced by TPM2\_PolicySigned() and TPM2\_PolicySecret() when the authorization has an expiration time. If *nonceTPM* was provided in the policy command, t*he* ticket is computed by

**HMAC**contextAlg(proof, (TPM\_ST\_AUTH\_xxx || cpHash || policyRef || authName

|| timeout || [timeEpoch] || [resetCount])) 1

where

**HMAC**contextAlg() an HMAC using the context integrity hash

proof a TPM secret value associated with the hierarchy of the object associated with authName

TPM\_ST\_AUTH\_xxx either TPM\_ST\_AUTH\_SIGNED or TPM\_ST\_AUTH\_SECRET; used to ensure that the ticket is properly used

cpHash optional hash of the authorized command

policyRef optional reference to a policy value

authName Name of the object that signed the authorization

timeout implementation-specific value indicating when the authorization expires

timeEpoch implementation-specific representation of the timeEpoch at the time the ticket was created

NOTE 1 Not included if *timeout* is zero.

resetCount implementation-specific representation of the TPM’s totalResetCount

NOTE 2 Not included it *timeout* is zeroor if *nonceTPM* was include in the authorization.

Table 99 — Definition of TPMT\_TK\_AUTH Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| tag {TPM\_ST\_AUTH\_SIGNED, TPM\_ST\_AUTH\_SECRET} | TPM\_ST | ticket structure tag |
| #TPM\_RC\_TAG |  | error returned when *tag* is not TPM\_ST\_AUTH |
| hierarchy | TPMI\_RH\_HIERARCHY+ | the hierarchy of the object used to produce the ticket |
| digest | TPM2B\_DIGEST | This shall be the HMAC produced using a proof value of *hierarchy*. |

EXAMPLE A NULL Auth Ticket is the tuple <TPM\_ST\_AUTH\_SIGNED, TPM\_RH\_NULL, 0x0000> or the tuple <TPM\_ST\_AUTH\_SIGNED, TPM\_RH\_NULL, 0x0000>

### TPMT\_TK\_HASHCHECK

This ticket is produced by TPM2\_SequenceComplete() or TPM2\_Hash() when the message that was digested did not start with TPM\_GENERATED\_VALUE. The ticket is computed by

**HMAC**contexAlg(proof, (TPM\_ST\_HASHCHECK || digest)) 1

where

**HMAC**contexAlg () an HMAC using the context integrity hash

proof a TPM secret value associated with the hierarchy indicated by the command

TPM\_ST\_HASHCHECK a value used to ensure that the ticket is properly used

digest the digest of the data

Table 100 — Definition of TPMT\_TK\_HASHCHECK Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| tag {TPM\_ST\_HASHCHECK} | TPM\_ST | ticket structure tag |
| #TPM\_RC\_TAG |  | error returned when is not TPM\_ST\_HASHCHECK |
| hierarchy | TPMI\_RH\_HIERARCHY+ | the hierarchy |
| digest | TPM2B\_DIGEST | This shall be the HMAC produced using a proof value of *hierarchy*. |

## Property Structures

### TPMS\_ALG\_PROPERTY

This structure is used to report the properties of an algorithm identifier. It is returned in response to a TPM2\_GetCapability() with *capability* = TPM\_CAP\_ALG.

Table 101 — Definition of TPMS\_ALG\_PROPERTY Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| alg | TPM\_ALG\_ID | an algorithm identifier |
| algProperties | TPMA\_ALGORITHM | the attributes of the algorithm |

### TPMS\_TAGGED\_PROPERTY

This structure is used to report the properties that are UINT32 values. It is returned in response to a TPM2\_GetCapability().

Table 102 — Definition of TPMS\_TAGGED\_PROPERTY Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| property | TPM\_PT | a property identifier |
| value | UINT32 | the value of the property |

### TPMS\_TAGGED\_PCR\_SELECT

This structure is used in TPM2\_GetCapability() to return the attributes of the PCR.

Table 103 — Definition of TPMS\_TAGGED\_PCR\_SELECT Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| tag | TPM\_PT\_PCR | the property identifier |
| sizeofSelect {PCR\_SELECT\_MIN:} | UINT8 | the size in octets of the *pcrSelect* array |
| pcrSelect [sizeofSelect] {:PCR\_SELECT\_MAX} | BYTE | the bit map of PCR with the identified property |

### TPMS\_TAGGED\_POLICY

This structure is used in TPM2\_GetCapability() to return the policy associated with a permanent handle.

Table 104 — Definition of TPMS\_TAGGED\_POLICY Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| handle | TPM\_HANDLE | a permanent handle |
| policyHash | TPMT\_HA | the policy algorithm and hash |

### TPMS\_ACT\_DATA

This structure is used in TPM2\_GetCapability() to return the ACT data.

Table 105 — Definition of TPMS\_ACT\_DATA Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| handle | TPM\_HANDLE | a permanent handle |
| timeout | UINT32 | the current timeout of the ACT |
| attributes | TPMA\_ACT | the state of the ACT |

## Lists

### TPML\_CC

A list of command codes may be input to the TPM or returned by the TPM depending on the command.

Table 106 — Definition of TPML\_CC Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| count | UINT32 | number of commands in the *commandCode* list; may be 0 |
| commandCodes[count]{:MAX\_CAP\_CC} | TPM\_CC | a list of command codes  The maximum only applies to a command code list in a command. The response size is limited only by the size of the parameter buffer. |
| #TPM\_RC\_SIZE |  | response code when count is greater than the maximum allowed list size |

### TPML\_CCA

This list is only used in TPM2\_GetCapability(capability = TPM\_CAP\_COMMANDS).

The values in the list are returned in TPMA\_CC->*commandIndex* order (see Table 37) with vendor-specific commands returned after other commands. Because of the other attributes, the commands may not be returned in strict numerical order.

Table 107 — Definition of TPML\_CCA Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| count | UINT32 | number of values in the *commandAttributes* list; may be 0 |
| commandAttributes[count]{:MAX\_CAP\_CC} | TPMA\_CC | a list of command codes attributes |

### TPML\_ALG

This list is returned by TPM2\_IncrementalSelfTest().

Table 108 — Definition of TPML\_ALG Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| count | UINT32 | number of algorithms in the *algorithms* list; may be 0 |
| algorithms[count]{:MAX\_ALG\_LIST\_SIZE} | TPM\_ALG\_ID | a list of algorithm IDs  The maximum only applies to an algorithm list in a command. The response size is limited only by the size of the parameter buffer. |
| #TPM\_RC\_SIZE |  | response code when *count* is greater than the maximum allowed list size |

### TPML\_HANDLE

This structure is used when the TPM returns a list of loaded handles when the *capability* in TPM2\_GetCapability() is TPM\_CAP\_HANDLE.

NOTE 1 MAX\_CAP\_HANDLES = (MAX\_CAP\_DATA / sizeof(TPM\_HANDLE))

NOTE 2 This list is not used as input to the TPM.

Table 109 — Definition of TPML\_HANDLE Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| count | UINT32 | the number of handles in the list  may have a value of 0 |
| handle[count]{: MAX\_CAP\_HANDLES} | TPM\_HANDLE | an array of handles |
| #TPM\_RC\_SIZE |  | response code when *count* is greater than the maximum allowed list size |

### TPML\_DIGEST

This list is used to convey a list of digest values. This type is used in TPM2\_PolicyOR() and in TPM2\_PCR\_Read().

Table 110 — Definition of TPML\_DIGEST Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| count {2:} | UINT32 | number of digests in the list, minimum is two for TPM2\_PolicyOR(). |
| digests[count]{:8} | TPM2B\_DIGEST | a list of digests  For TPM2\_PolicyOR(), all digests will have been computed using the digest of the policy session. For TPM2\_PCR\_Read(), each digest will be the size of the digest for the bank containing the PCR. |
| #TPM\_RC\_SIZE |  | response code when *count* is not at least two or is greater than eight |

### TPML\_DIGEST\_VALUES

This list is used to convey a list of digest values. This type is returned by TPM2\_PCR\_Event() and TPM2\_EventSequenceComplete() and is an input for TPM2\_PCR\_Extend().

NOTE 1 This construct limits the number of hashes in the list to the number of digests implemented in the TPM rather than the number of PCR banks. This allows extra values to appear in a call to TPM2\_PCR\_Extend().

NOTE 2 The digest for an unimplemented hash algorithm may not be in a list because the TPM may not recognize the algorithm as being a hash and it may not know the digest size.

Table 111 — Definition of TPML\_DIGEST\_VALUES Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| count | UINT32 | number of digests in the list |
| digests[count]{:HASH\_COUNT} | TPMT\_HA | a list of tagged digests |
| #TPM\_RC\_SIZE |  | response code when *count* is greater than the possible number of banks |

### TPML\_PCR\_SELECTION

This list is used to indicate the PCR that are included in a selection when more than one PCR value may be selected.

This structure is an input parameter to TPM2\_PolicyPCR() to indicate the PCR that will be included in the digest of PCR for the authorization. The structure is used in TPM2\_PCR\_Read() command to indicate the PCR values to be returned and in the response to indicate which PCR are included in the list of returned digests. The structure is an output parameter from TPM2\_Create() and indicates the PCR used in the digest of the PCR state when the object was created. The structure is also contained in the attestation structure of TPM2\_Quote().

When this structure is used to select PCR to be included in a digest, the selected PCR are concatenated to create a “message” containing all of the PCR, and then the message is hashed using the context-specific hash algorithm.

Table 112 — Definition of TPML\_PCR\_SELECTION Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| count | UINT32 | number of selection structures  A value of zero is allowed. |
| pcrSelections[count]{:HASH\_COUNT} | TPMS\_PCR\_SELECTION | list of selections |
| #TPM\_RC\_SIZE |  | response code when *count* is greater than the possible number of banks |

### TPML\_ALG\_PROPERTY

This list is used to report on a list of algorithm attributes. It is returned in a TPM2\_GetCapability().

NOTE MAX\_CAP\_ALGS = MAX\_CAP\_DATA / sizeof(TPMS\_ALG\_PROPERTY).

Table 113 — Definition of TPML\_ALG\_PROPERTY Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| count | UINT32 | number of algorithm properties structures  A value of zero is allowed. |
| algProperties[count]{:MAX\_CAP\_ALGS} | TPMS\_ALG\_PROPERTY | list of properties |

### TPML\_TAGGED\_TPM\_PROPERTY

This list is used to report on a list of properties that are TPMS\_TAGGED\_PROPERTY values. It is returned by a TPM2\_GetCapability().

NOTE MAX\_TPM\_PROPERTIES = MAX\_CAP\_DATA / sizeof(TPMS\_TAGGED\_PROPERTY).

Table 114 — Definition of TPML\_TAGGED\_TPM\_PROPERTY Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| count | UINT32 | number of properties  A value of zero is allowed. |
| tpmProperty[count]{:MAX\_TPM\_PROPERTIES} | TPMS\_TAGGED\_PROPERTY | an array of tagged properties |

### TPML\_TAGGED\_PCR\_PROPERTY

This list is used to report on a list of properties that are TPMS\_PCR\_SELECT values. It is returned by a TPM2\_GetCapability().

NOTE MAX\_PCR\_PROPERTIES = MAX\_CAP\_DATA / sizeof(TPMS\_TAGGED\_PCR\_SELECT).

Table 115 — Definition of TPML\_TAGGED\_PCR\_PROPERTY Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| count | UINT32 | number of properties  A value of zero is allowed. |
| pcrProperty[count]{:MAX\_PCR\_PROPERTIES} | TPMS\_TAGGED\_PCR\_SELECT | a tagged PCR selection |

### TPML\_ECC\_CURVE

This list is used to report the ECC curve ID values supported by the TPM. It is returned by a TPM2\_GetCapability().

NOTE MAX\_ECC\_CURVES = MAX\_CAP\_DATA / sizeof(TPM\_ECC\_CURVE).

Table 116 — Definition of {ECC} TPML\_ECC\_CURVE Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| count | UINT32 | number of curves  A value of zero is allowed. |
| eccCurves[count]{:MAX\_ECC\_CURVES} | TPM\_ECC\_CURVE | array of ECC curve identifiers |

### TPML\_TAGGED\_POLICY

This list is used to report the authorization policy values for permanent handles. This is list may be generated by TPM2\_GetCapabiltiy(). A permanent handle that cannot have a policy is not included in the list.

NOTE MAX\_TAGGED\_POLICIES = MAX\_CAP\_DATA / sizeof(TPMS\_TAGGED\_POLICY).

Table 117 — Definition of TPML\_TAGGED\_POLICY Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| count | UINT32 | number of tagged policies  A value of zero is allowed. |
| policies[count]{:MAX\_TAGGED\_POLICIES} | TPMS\_TAGGED\_POLICY | array of tagged policies |

### TPML\_ACT\_DATA

This list is used to report the timeout and state for the ACT. This list may be generated by TPM2\_GetCapabilty(). Only implemented ACT are present in the list

NOTE MAX\_ACT\_DATA = MAX\_CAP\_DATA / sizeof(TPMS\_ACT\_DATA).

Table 118 — Definition of TPML\_ACT\_DATA Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| count | UINT32 | number of ACT instances  A value of zero is allowed. |
| actData[count]{:MAX\_ACT\_DATA} | TPMS\_ACT\_DATA | array of ACT data |

## Capabilities Structures

It is required that each parameter in this union be a list (TPML).

The number of returned elements in each list is determined by the size of each list element and the maximum size set by the vendor as the capability buffer (MAX\_CAP\_BUFFER in TPM\_PT\_MAX\_CAP\_BUFFER). The maximum number of bytes in a list is:

MAX\_CAP\_DATA = (MAX\_CAP\_BUFFER – sizeof(TPM\_CAP) – sizeof(UINT32) 1

The maximum number of entries is then the number of complete list elements that will fit in MAX\_CAP\_DATA.

EXAMPLE For a 1024-octet MAX\_CAP\_BUFFER a response containing a TPML\_HANDLE could have (1024 - 4 – 4) / 4 = 254 handles.

### TPMU\_CAPABILITIES

Table 119 — Definition of TPMU\_CAPABILITIES Union <OUT>

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| algorithms | TPML\_ALG\_PROPERTY | TPM\_CAP\_ALGS |  |
| handles | TPML\_HANDLE | TPM\_CAP\_HANDLES |  |
| command | TPML\_CCA | TPM\_CAP\_COMMANDS |  |
| ppCommands | TPML\_CC | TPM\_CAP\_PP\_COMMANDS |  |
| auditCommands | TPML\_CC | TPM\_CAP\_AUDIT\_COMMANDS |  |
| assignedPCR | TPML\_PCR\_SELECTION | TPM\_CAP\_PCRS |  |
| tpmProperties | TPML\_TAGGED\_TPM\_PROPERTY | TPM\_CAP\_TPM\_PROPERTIES |  |
| pcrProperties | TPML\_TAGGED\_PCR\_PROPERTY | TPM\_CAP\_PCR\_PROPERTIES |  |
| eccCurves | TPML\_ECC\_CURVE | TPM\_CAP\_ECC\_CURVES | TPM\_ALG\_ECC |
| authPolicies | TPML\_TAGGED\_POLICY | TPM\_CAP\_AUTH\_POLICIES |  |
| actData | TPML\_ACT\_DATA | TPM\_CAP\_ACT |  |

### TPMS\_CAPABILITY\_DATA

This data area is returned in response to a TPM2\_GetCapability().

Table 120 — Definition of TPMS\_CAPABILITY\_DATA Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| capability | TPM\_CAP | the capability |
| [capability]data | TPMU\_CAPABILITIES | the capability data |

## Clock/Counter Structures

### TPMS\_CLOCK\_INFO

This structure is used in each of the attestation commands.

Table 121 — Definition of TPMS\_CLOCK\_INFO Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| clock | UINT64 | time value in milliseconds that advances while the TPM is powered  NOTE The interpretation of the time-origin (*clock*=0) is out of the scope of this specification, although Coordinated Universal Time (UTC) is expected to be a common convention. This structure element is used to report on the TPM's Clock value.  This value is reset to zero when the Storage Primary Seed is changed (TPM2\_Clear()).  This value may be advanced by TPM2\_ClockSet(). |
| resetCount | UINT32 | number of occurrences of TPM Reset since the last TPM2\_Clear() |
| restartCount | UINT32 | number of times that TPM2\_Shutdown() or \_TPM\_Hash\_Start have occurred since the last TPM Reset or TPM2\_Clear(). |
| safe | TPMI\_YES\_NO | no value of *Clock* greater than the current value of *Clock* has been previously reported by the TPM. Set to YES on TPM2\_Clear(). |

### *Clock*

*Clock* is a monotonically increasing counter that advances whenever power is applied to the TPM. The value of *Clock* may be set forward with TPM2\_ClockSet() if Owner Authorization or Platform Authorization is provided. The value of *Clock* is incremented each millisecond.

TPM2\_Clear() will set *Clock* to zero.

*Clock* will be non-volatile but may have a volatile component that is updated every millisecond with the non-volatile component updated at a lower rate. The reference for the millisecond timer is the TPM oscillator. If the implementation uses a volatile component, the non-volatile component shall be updated no less frequently than every 222 milliseconds (~69.9 minutes). The update rate of the non-volatile portion of *Clock* shall be reported by a TPM2\_GetCapability() with *capability* = TPM\_CAP\_TPM\_PROPERTIES and *property* = TPM\_PT\_CLOCK\_UPDATE.

### *ResetCount*

This counter shall increment on each TPM Reset. This counter shall be reset to zero by TPM2\_Clear().

### *RestartCount*

This counter shall increment by one for each TPM Restart or TPM Resume. The *restartCount* shall be reset to zero on a TPM Reset or TPM2\_Clear().

### *Safe*

This parameter is set to YES when the value reported in *Clock* is guaranteed to be greater than any previous value for the current Owner. It is set to NO when the value of *Clock* may have been reported in a previous attestation or access.

EXAMPLE 1 If *Safe* was NO at TPM2\_Shutdown() and *Clock* does not update unless a command is received, *Safe* will be NO if a TPM2\_Startup() was preceded by TPM2\_Shutdown() with no intervening commands. If *Clock* updates independent of commands, the non-volatile bits of *Clock* can be updated, so *Safe* can be YES at TPM2\_Startup().

EXAMPLE 2 This parameter will be YES after the non-volatile bits of *Clock* have been updated at the end of an update interval.

If a TPM implementation does not implement *Clock*, *Safe* shall always be NO and TPMS\_CLOCK\_INFO.*clock* shall always be zero.

This parameter will be set to YES by TPM2\_Clear().

### TPMS\_TIME\_INFO

This structure is used in, e.g., the TPM2\_GetTime() attestation and TPM2\_ReadClock().

The *Time* value reported in this structure is reset whenever power to the *Time* circuit is reestablished. If required, an implementation may reset the value of *Time* any time before the TPM returns after TPM2\_Startup(). The value of *Time* shall increment continuously while power is applied to the TPM.

Table 122 — Definition of TPMS\_TIME\_INFO Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| time | UINT64 | time in milliseconds since the *TIme* circuit was last reset  This structure element is used to report on the TPM's *Time* value. |
| clockInfo | TPMS\_CLOCK\_INFO | a structure containing the clock information |

## TPM Attestation Structures

### Introduction

Clause 10.12 describes the structures that are used when a TPM creates a structure to be signed. The signing structures follow a standard format TPM2B\_ATTEST with case-specific information embedded.

### TPMS\_TIME\_ATTEST\_INFO

This structure is used when the TPM performs TPM2\_GetTime.

Table 123 — Definition of TPMS\_TIME\_ATTEST\_INFO Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| time | TPMS\_TIME\_INFO | the *Time*, *Clock*, *resetCount*, *restartCount*, and *Safe* indicator |
| firmwareVersion | UINT64 | a TPM vendor-specific value indicating the version number of the firmware |

### TPMS\_CERTIFY\_INFO

This is the attested data for TPM2\_Certify().

Table 124 — Definition of TPMS\_CERTIFY\_INFO Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| name | TPM2B\_NAME | Name of the certified object |
| qualifiedName | TPM2B\_NAME | Qualified Name of the certified object |

### TPMS\_QUOTE\_INFO

This is the *attested* data for TPM2\_Quote().

Table 125 — Definition of TPMS\_QUOTE\_INFO Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| pcrSelect | TPML\_PCR\_SELECTION | information on *algID*, PCR selected and digest |
| pcrDigest | TPM2B\_DIGEST | digest of the selected PCR using the hash of the signing key |

### TPMS\_COMMAND\_AUDIT\_INFO

This is the *attested* data for TPM2\_GetCommandAuditDigest().

Table 126 — Definition of TPMS\_COMMAND\_AUDIT\_INFO Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| auditCounter | UINT64 | the monotonic audit counter |
| digestAlg | TPM\_ALG\_ID | hash algorithm used for the command audit |
| auditDigest | TPM2B\_DIGEST | the current value of the audit digest |
| commandDigest | TPM2B\_DIGEST | digest of the command codes being audited using *digestAlg* |

### TPMS\_SESSION\_AUDIT\_INFO

This is the *attested* data for TPM2\_GetSessionAuditDigest().

Table 127 — Definition of TPMS\_SESSION\_AUDIT\_INFO Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| exclusiveSession | TPMI\_YES\_NO | current exclusive status of the session  TRUE if all of the commands recorded in the *sessionDigest* were executed without any intervening TPM command that did not use this audit session |
| sessionDigest | TPM2B\_DIGEST | the current value of the session audit digest |

### TPMS\_CREATION\_INFO

This is the *attested* data for TPM2\_CertifyCreation().

Table 128 — Definition of TPMS\_CREATION\_INFO Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| objectName | TPM2B\_NAME | Name of the object |
| creationHash | TPM2B\_DIGEST | creationHash |

### TPMS\_NV\_CERTIFY\_INFO

This structure contains the Name and contents of the selected NV Index that is certified by TPM2\_NV\_Certify().

Table 129 — Definition of TPMS\_NV\_CERTIFY\_INFO Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| indexName | TPM2B\_NAME | Name of the NV Index |
| offset | UINT16 | the *offset* parameter of TPM2\_NV\_Certify() |
| nvContents | TPM2B\_MAX\_NV\_BUFFER | contents of the NV Index |

### TPMS\_NV\_DIGEST\_CERTIFY\_INFO

This structure contains the Name and hash of the contents of the selected NV Index that is certified by TPM2\_NV\_Certify(). The data is hashed using hash of the signing scheme.

NOTE This structure was added in revision 01.53 to support alternate TPM2\_NV\_Certify() behavior.

Table 130 — Definition of TPMS\_NV\_DIGEST\_CERTIFY\_INFO Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| indexName | TPM2B\_NAME | Name of the NV Index |
| nvDigest | TPM2B\_DIGEST | hash of the contents of the index |

### TPMI\_ST\_ATTEST

Table 131 — Definition of (TPM\_ST) TPMI\_ST\_ATTEST Type <OUT>

| Value | Description |
| --- | --- |
| TPM\_ST\_ATTEST\_CERTIFY | generated by TPM2\_Certify() |
| TPM\_ST\_ATTEST\_QUOTE | generated by TPM2\_Quote() |
| TPM\_ST\_ATTEST\_SESSION\_AUDIT | generated by TPM2\_GetSessionAuditDigest() |
| TPM\_ST\_ATTEST\_COMMAND\_AUDIT | generated by TPM2\_GetCommandAuditDigest() |
| TPM\_ST\_ATTEST\_TIME | generated by TPM2\_GetTime() |
| TPM\_ST\_ATTEST\_CREATION | generated by TPM2\_CertifyCreation() |
| TPM\_ST\_ATTEST\_NV | generated by TPM2\_NV\_Certify() |
| TPM\_ST\_ATTEST\_NV\_DIGEST | generated by TPM2\_NV\_Certify() |

### TPMU\_ATTEST

Table 132 — Definition of TPMU\_ATTEST Union <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Selector |
| certify | TPMS\_CERTIFY\_INFO | TPM\_ST\_ATTEST\_CERTIFY |
| creation | TPMS\_CREATION\_INFO | TPM\_ST\_ATTEST\_CREATION |
| quote | TPMS\_QUOTE\_INFO | TPM\_ST\_ATTEST\_QUOTE |
| commandAudit | TPMS\_COMMAND\_AUDIT\_INFO | TPM\_ST\_ATTEST\_COMMAND\_AUDIT |
| sessionAudit | TPMS\_SESSION\_AUDIT\_INFO | TPM\_ST\_ATTEST\_SESSION\_AUDIT |
| time | TPMS\_TIME\_ATTEST\_INFO | TPM\_ST\_ATTEST\_TIME |
| nv | TPMS\_NV\_CERTIFY\_INFO | TPM\_ST\_ATTEST\_NV |
| nvDigest | TPMS\_NV\_DIGEST\_CERTIFY\_INFO | TPM\_ST\_ATTEST\_NV\_DIGEST |

### TPMS\_ATTEST

This structure is used on each TPM-generated signed structure. The signature is over this structure.

When the structure is signed by a key in the Storage hierarchy, the values of *clockInfo.resetCount*, *clockInfo.restartCount*, and *firmwareVersion* are obfuscated with a per-key obfuscation value.

Table 133 — Definition of TPMS\_ATTEST Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| magic | TPM\_GENERATED | the indication that this structure was created by a TPM (always TPM\_GENERATED\_VALUE) |
| type | TPMI\_ST\_ATTEST | type of the attestation structure |
| qualifiedSigner | TPM2B\_NAME | Qualified Name of the signing key |
| extraData | TPM2B\_DATA | external information supplied by caller  NOTE A TPM2B\_DATA structure provides room for a digest and a method indicator to indicate the components of the digest. The definition of this method indicator is outside the scope of this specification. |
| clockInfo | TPMS\_CLOCK\_INFO | Clock, resetCount, restartCount, and Safe |
| firmwareVersion | UINT64 | TPM-vendor-specific value identifying the version number of the firmware |
| [type]attested | TPMU\_ATTEST | the type-specific attestation information |

### TPM2B\_ATTEST

This sized buffer to contain the signed structure. The *attestationData* is the signed portion of the structure. The *size* parameter is not signed.

Table 134 — Definition of TPM2B\_ATTEST Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size of the *attestationData* structure |
| attestationData[size]{:sizeof(TPMS\_ATTEST)} | BYTE | the signed structure |

## Authorization Structures

### Introduction

The structures in 10.13 are used for all authorizations. One or more of these structures will be present in a command or response that has a tag of TPM\_ST\_SESSIONS.

### TPMS\_AUTH\_COMMAND

This is the format used for each of the authorizations in the session area of a command.

Table 135 — Definition of TPMS\_AUTH\_COMMAND Structure <IN>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| sessionHandle | TPMI\_SH\_AUTH\_SESSION+ | the session handle |
| nonce | TPM2B\_NONCE | the session nonce, may be the Empty Buffer |
| sessionAttributes | TPMA\_SESSION | the session attributes |
| hmac | TPM2B\_AUTH | either an HMAC, a password, or an EmptyAuth |

### TPMS\_AUTH\_RESPONSE

This is the format for each of the authorizations in the session area of the response. If the TPM returns TPM\_RC\_SUCCESS, then the session area of the response contains the same number of authorizations as the command and the authorizations are in the same order.

Table 136 — Definition of TPMS\_AUTH\_RESPONSE Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| nonce | TPM2B\_NONCE | the session nonce, may be the Empty Buffer |
| sessionAttributes | TPMA\_SESSION | the session attributes |
| hmac | TPM2B\_AUTH | either an HMAC or an EmptyAuth |

# Algorithm Parameters and Structures

## Symmetric

### Introduction

Clause 11.1 defines the parameters and structures for describing symmetric algorithms.

### TPMI\_!ALG.S\_KEY\_BITS

This interface type defines the supported key sizes for a symmetric algorithm. This type is used to allow the unmarshaling routine to generate the proper validation code for the supported key sizes. An implementation that supports different key sizes would have a different set of selections.

Each implemented algorithm would have a value for the implemented key sizes for that implemented algorithm. That value would have a name in the form !ALG\_KEY\_SIZES\_BITS where “!ALG” would represent the characteristic name of the algorithm (such as “AES).

NOTE 1 Key size is expressed in bits.

Table 137 — Definition of {!ALG.S} (TPM\_KEY\_BITS) TPMI\_!ALG.S\_KEY\_BITS Type

|  |  |
| --- | --- |
| Parameter | Description |
| $!ALG.S\_KEY\_SIZES\_BITS | number of bits in the key |
| #TPM\_RC\_VALUE | error when key size is not supported |

### TPMU\_SYM\_KEY\_BITS

This union is used to collect the symmetric encryption key sizes.

The *xor* entry is a hash algorithms selector and not a key size in bits. This overload is used in order to avoid an additional level of indirection with another union and another set of selectors.

The *xor* entry is only selected in a TPMT\_SYM\_DEF, which is used to select the parameter encryption value.

Table 138 — Definition of TPMU\_SYM\_KEY\_BITS Union

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| !ALG.S | TPMI\_!ALG.S\_KEY\_BITS | TPM\_ALG\_!ALG.S | all symmetric algorithms |
| sym | TPM\_KEY\_BITS |  | this entry is used by the reference code to refer to the key bits field in a way that is independent of the symmetric algorithm |
| xor | TPMI\_ALG\_HASH | TPM\_ALG\_XOR | overload for using *xor*  NOTE TPM\_ALG\_NULL is not allowed |
| null |  | TPM\_ALG\_NULL |  |

### TPMU\_SYM\_MODE

This is the union of all modes for all symmetric algorithms.

NOTE This union definition allows the mode value in a TPMT\_SYM\_DEF to be empty when the selector is TPM\_ALG\_XOR because the XOR algorithm does not have a mode.

Table 139 — Definition of TPMU\_SYM\_MODE Union

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| !ALG.S | TPMI\_ALG\_SYM\_MODE+ | TPM\_ALG\_!ALG.S |  |
| sym | TPMI\_ALG\_SYM\_MODE+ |  | this entry is used by the reference code to refer to the mode field in a way that is independent of the symmetric algorithm |
| xor |  | TPM\_ALG\_XOR | no mode selector |
| null |  | TPM\_ALG\_NULL | no mode selector |

### TPMU\_SYM\_DETAILS

This union allows additional parameters to be added for a symmetric cipher. Currently, no additional parameters are required for any of the symmetric algorithms.

NOTE The “x” character in the table title will suppress generation of this type as the parser is not, at this time, able to generate the proper values (a union of all empty data types). When an algorithm is added that requires additional parameterization, the Type column will contain a value and the “x” may be removed.

Table 140 —xDefinition of TPMU\_SYM\_DETAILS Union

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| !ALG.S |  | TPM\_ALG\_!ALG |  |
| sym |  |  | this entry is used by the reference code to refer to the details field in a way that is independent of the symmetric algorithm |
| xor |  | TPM\_ALG\_XOR |  |
| null |  | TPM\_ALG\_NULL |  |

### TPMT\_SYM\_DEF

The TPMT\_SYM\_DEF structure is used to select an algorithm to be used for parameter encryption in those cases when different symmetric algorithms may be selected.

Table 141 — Definition of TPMT\_SYM\_DEF Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| algorithm | +TPMI\_ALG\_SYM | indicates a symmetric algorithm |
| [algorithm]keyBits | TPMU\_SYM\_KEY\_BITS | a supported key size |
| [algorithm]mode | TPMU\_SYM\_MODE | the mode for the key |
| //[algorithm]details | TPMU\_SYM\_DETAILS | contains additional algorithm details  NOTE This is commented out at this time as the parser may not produce the proper code for a union if none of the selectors produces any data. |

### TPMT\_SYM\_DEF\_OBJECT

This structure is used when different symmetric block cipher (not XOR) algorithms may be selected. If the Object can be an ordinary parent (not a derivation parent), this must be the first field in the Object's parameter (see 12.2.3.7) field.

Table 142 — Definition of TPMT\_SYM\_DEF\_OBJECT Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| algorithm | +TPMI\_ALG\_SYM\_OBJECT | selects a symmetric block cipher  When used in the parameter area of a parent object, this shall be a supported block cipher and not TPM\_ALG\_NULL |
| [algorithm]keyBits | TPMU\_SYM\_KEY\_BITS | the key size |
| [algorithm]mode | TPMU\_SYM\_MODE | default mode  When used in the parameter area of a parent object, this shall be TPM\_ALG\_CFB. |
| //[algorithm]details | TPMU\_SYM\_DETAILS | contains the additional algorithm details, if any  NOTE This is commented out at this time as the parser may not produce the proper code for a union if none of the selectors produces any data. |

### TPM2B\_SYM\_KEY

This structure is used to hold a symmetric key in the sensitive area of an asymmetric object.

The number of bits in the key is in *keyBits* in the public area. When *keyBits* is not an even multiple of 8 bits, the unused bits of *buffer* will be the most significant bits of *buffer*[0] and *size* will be rounded up to the number of octets required to hold all bits of the key.

NOTE MAX\_SYM\_KEY\_BYTES will be the larger of the largest symmetric key supported by the TPM and the largest digest produced by any hashing algorithm implemented on the TPM.

Table 143 — Definition of TPM2B\_SYM\_KEY Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size, in octets, of the buffer containing the key; may be zero |
| buffer [size] {:MAX\_SYM\_KEY\_BYTES} | BYTE | the key |

### TPMS\_SYMCIPHER\_PARMS

This structure contains the parameters for a symmetric block cipher object.

Table 144 — Definition of TPMS\_SYMCIPHER\_PARMS Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| sym | TPMT\_SYM\_DEF\_OBJECT | a symmetric block cipher |

### TPM2B\_LABEL

This buffer holds a *label* or *context* value. For interoperability and backwards compatibility, LABEL\_MAX\_BUFFER is the minimum of the largest digest on the device and the largest ECC parameter (MAX\_ECC\_KEY\_BYTES) but no more than 32 bytes.

All implementations are required to support at least one hash algorithm that produces a digest of 32 bytes or larger; and any implementation that supports ECC is required to support at least one curve with a key size of 32-bytes or larger.

NOTE Although the maximum size allowed for a *label* or *context* is 32 bytes, the object data structure needs to be sized to allow a 32-byte value.

Table 145 — Definition of TPM2B\_LABEL Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 |  |
| buffer[size]{:LABEL\_MAX\_BUFFER} | BYTE | symmetric data for a created object or the *label* and *context* for a derived object |

### TPMS\_DERIVE

This structure contains the *label* and *context* fields for a derived object. These values are used in the derivation KDF. The values in the *unique* field of *inPubli*c area template take precedence over the values in the *inSensitive* parameter.

Table 146 — Definition of TPMS\_DERIVE Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| label | TPM2B\_LABEL |  |
| context | TPM2B\_LABEL |  |

### TPM2B\_DERIVE

Table 147 — Definition of TPM2B\_DERIVE Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 |  |
| buffer[size]{: sizeof(TPMS\_DERIVE)} | BYTE | symmetric data for a created object or the *label* and *context* for a derived object |

### TPMU\_SENSITIVE\_CREATE

This structure allows a TPM2B\_SENSITIVE\_CREATE structure to carry either a TPM2B\_SENSITVE\_DATA or a TPM2B\_DERIVE structure. The contents of the union are determined by context. When an object is being derived, the derivation values are present.

For interoperability, MAX\_SYM\_DATA should be 128.

NOTE No marshaling code is automatically generated for this union as it has no selectors that would allow the code to know the context and which member to unmarshal.

Table 148 — Definition of TPMU\_SENSITIVE\_CREATE Union <>

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| create[MAX\_SYM\_DATA] | BYTE |  | sensitive data for a created symmetric Object |
| derive | TPMS\_DERIVE |  | *label* and *context* for a derived Object |

### TPM2B\_SENSITIVE\_DATA

This buffer wraps the TPMU\_SENSITIVE\_CREATE structure.

Table 149 — Definition of TPM2B\_SENSITIVE\_DATA Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 |  |
| buffer[size]{: sizeof(TPMU\_SENSITIVE\_CREATE)} | BYTE | symmetric data for a created object or the *label* and *context* for a derived object |

### TPMS\_SENSITIVE\_CREATE

This structure defines the values to be placed in the sensitive area of a created object. This structure is only used within a TPM2B\_SENSITIVE\_CREATE structure.

NOTE When sent to the TPM or unsealed, data is usually encrypted using parameter encryption.

If *data.size* is not zero, and the object is not a *keyedHash*, *data.size* must match the size indicated in the *keySize* *o*f *public.parameters.*If the object is a *keyedHash*, *data*.*size* may be any value up to the maximum allowed in a TPM2B\_SENSITIVE\_DATA.

For an asymmetric object, data shall be an Empty Buffer and *sensitiveDataOrigin* shall be SET.

Table 150 — Definition of TPMS\_SENSITIVE\_CREATE Structure <IN>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| userAuth | TPM2B\_AUTH | the USER auth secret value |
| data | TPM2B\_SENSITIVE\_DATA | data to be sealed, a key, or derivation values |

### TPM2B\_SENSITIVE\_CREATE

This structure contains the sensitive creation data in a sized buffer. This structure is defined so that both the *userAuth* and *data* values of the TPMS\_SENSITIVE\_CREATE may be passed as a single parameter for parameter encryption purposes.

Table 151 — Definition of TPM2B\_SENSITIVE\_CREATE Structure <IN, S>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size= | UINT16 | size of *sensitive* in octets (may not be zero)  NOTE The *userAuth* and data parameters in this buffer may both be zero length but the minimum size of this parameter will be the sum of the size fields of the two parameters of the TPMS\_SENSITIVE\_CREATE. |
| sensitive | TPMS\_SENSITIVE\_CREATE | data to be sealed or a symmetric key value. |

### TPMS\_SCHEME\_HASH

This structure is the scheme data for schemes that only require a hash to complete their definition.

Table 152 — Definition of TPMS\_SCHEME\_HASH Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| hashAlg | TPMI\_ALG\_HASH | the hash algorithm used to digest the message |

### TPMS\_SCHEME\_ECDAA

This definition is for split signing schemes that require a commit count.

Table 153 — Definition of {ECC} TPMS\_SCHEME\_ECDAA Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| hashAlg | TPMI\_ALG\_HASH | the hash algorithm used to digest the message |
| count | UINT16 | the counter value that is used between TPM2\_Commit() and the sign operation |

### TPMI\_ALG\_KEYEDHASH\_SCHEME

This is the list of values that may appear in a *keyedHash* as the *scheme* parameter.

Table 154 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_KEYEDHASH\_SCHEME Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_HMAC | the "signing" scheme |
| TPM\_ALG\_XOR | the "obfuscation" scheme |
| +TPM\_ALG\_NULL |  |
| #TPM\_RC\_VALUE |  |

### HMAC\_SIG\_SCHEME

Table 155 — Definition of Types for HMAC\_SIG\_SCHEME

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| TPMS\_SCHEME\_HASH | TPMS\_SCHEME\_HMAC |  |

### TPMS\_SCHEME\_XOR

This structure is for the XOR encryption scheme.

NOTE Prior to revision 01.47, the TPM\_ALG\_NULL hash algorithm was permitted. This produced a zero length key. The TPM\_ALG\_NULL *hashAlg* now returns TPM\_RC\_HASH.

Table 156 — Definition of TPMS\_SCHEME\_XOR Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| hashAlg | TPMI\_ALG\_HASH | the hash algorithm used to digest the message |
| kdf | TPMI\_ALG\_KDF+ | the key derivation function |

### TPMU\_SCHEME\_KEYEDHASH

Table 157 — Definition of TPMU\_SCHEME\_KEYEDHASH Union <IN/OUT >

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| hmac | TPMS\_SCHEME\_HMAC | TPM\_ALG\_HMAC | the "signing" scheme |
| xor | TPMS\_SCHEME\_XOR | TPM\_ALG\_XOR | the "obfuscation" scheme |
| null |  | TPM\_ALG\_NULL |  |

### TPMT\_KEYEDHASH\_SCHEME

This structure is used for a hash signing object.

Table 158 — Definition of TPMT\_KEYEDHASH\_SCHEME Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| scheme | +TPMI\_ALG\_KEYEDHASH\_SCHEME | selects the scheme |
| [scheme]details | TPMU\_SCHEME\_KEYEDHASH | the scheme parameters |

## Asymmetric

### Signing Schemes

#### Introduction

These structures are used to define the method in which the signature is to be created. These schemes would appear in an object’s public area and in commands where the signing scheme is variable.

Every scheme is required to indicate a hash that is used in digesting the message.

#### RSA Signature Schemes

These are the RSA schemes that only need a hash algorithm as a scheme parameter.

For the TPM\_ALG\_RSAPSS signing scheme, the same hash algorithm is used for digesting TPM-generated data (an attestation structure) and in the KDF used for the masking operation. The salt size is always the largest salt value that will fit into the available space.

Table 159 — Definition of {RSA} Types for RSA Signature Schemes

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| TPMS\_SCHEME\_HASH | TPMS\_SIG\_SCHEME\_!ALG.AX |  |

#### ECC Signature Schemes

Most of the ECC signature schemes only require a hash algorithm to complete the definition and can be typed as TPMS\_SCHEME\_HASH. Anonymous algorithms also require a count value so they are typed to be TPMS\_SCHEME\_ECDAA.

Table 160 — Definition of {ECC} Types for ECC Signature Schemes

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| TPMS\_SCHEME\_HASH | TPMS\_SIG\_SCHEME\_!ALG.AX | all asymmetric signing schemes |
| TPMS\_SCHEME\_ECDAA | TPMS\_SIG\_SCHEME\_!ALG.AXN | schemes that need a hash and a count |

#### TPMU\_SIG\_SCHEME

This is the union of all of the signature schemes.

NOTE The TPMS\_SIG\_SCHEME\_!ALG is determined by Table 159 or Table 160 and will be either a TPMS\_SCHEME\_HASH or a TPMS\_SCHEME\_ECDAA.

Table 161 — Definition of TPMU\_SIG\_SCHEME Union <IN/OUT >

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| !ALG.ax | TPMS\_SIG\_SCHEME\_!ALG | TPM\_ALG\_!ALG | all signing schemes including anonymous schemes |
| hmac | TPMS\_SCHEME\_HMAC | TPM\_ALG\_HMAC | the HMAC scheme |
| any | TPMS\_SCHEME\_HASH |  | selector that allows access to digest for any signing scheme |
| null |  | TPM\_ALG\_NULL | no scheme or default |

#### TPMT\_SIG\_SCHEME

Table 162 — Definition of TPMT\_SIG\_SCHEME Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| scheme | +TPMI\_ALG\_SIG\_SCHEME | scheme selector |
| [scheme]details | TPMU\_SIG\_SCHEME | scheme parameters |

### Encryption Schemes

#### Introduction

These structures are used to indicate the algorithm used for the encrypting process. These schemes would appear in an object’s public area.

NOTE With ECC, the only encryption is with a key exchange of a symmetric key or seed.

#### RSA Encryption Schemes

These are the RSA encryption schemes that only need a hash algorithm as a controlling parameter.

NOTE: These types do not appear in the reference code in the specification but are used in the unmarshaling code.

Table 163 — Definition of Types for {RSA} Encryption Schemes

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| TPMS\_SCHEME\_HASH | TPMS\_ENC\_SCHEME\_!ALG.AEH | schemes that only need a hash |
| TPMS\_EMPTY | TPMS\_ENC\_SCHEME\_!ALG.AE | schemes that need nothing |

#### ECC Key Exchange Schemes

These are the ECC schemes that only need a hash algorithm as a controlling parameter.

NOTE: These types do not appear in the reference code in the specification but are used in the unmarshaling code.

Table 164 — Definition of Types for {ECC} ECC Key Exchange

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| TPMS\_SCHEME\_HASH | TPMS\_KEY\_SCHEME\_!ALG.AM | schemes that need a hash |

### Key Derivation Schemes

#### Introduction

These structures are used to define the key derivation for symmetric secret sharing using asymmetric methods. A secret sharing scheme is required in any asymmetric key with the *decrypt* attribute SET.

These schemes would appear in an object’s public area and in commands where the secret sharing scheme is variable.

Each scheme includes a symmetric algorithm and a KDF selection.

The qualifying value for each of the KDF schemes is the hash algorithm.

NOTE: These types do not appear in the reference code in the specification but are used in the unmarshaling code.

Table 165 — Definition of Types for KDF Schemes

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| TPMS\_SCHEME\_HASH | TPMS\_KDF\_SCHEME\_!ALG.HM | hash-based key- or mask-generation functions |

#### TPMU\_KDF\_SCHEME

Table 166 — Definition of TPMU\_KDF\_SCHEME Union <IN/OUT>

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| !ALG.HM | TPMS\_KDF\_SCHEME\_!ALG.HM | TPM\_ALG\_!ALG.HM |  |
| null |  | TPM\_ALG\_NULL |  |

#### TPMT\_KDF\_SCHEME

Table 167 — Definition of TPMT\_KDF\_SCHEME Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| scheme | +TPMI\_ALG\_KDF | scheme selector |
| [scheme]details | TPMU\_KDF\_SCHEME | scheme parameters |

#### TPMI\_ALG\_ASYM\_SCHEME

List of all of the scheme types for any asymmetric algorithm.

NOTE 1 This is the selector value used to define TPMT\_ASYM\_SCHEME.

NOTE 2 Most tokens are exclusive in order to filter out SM2 and other multi-protocol algorithm identifiers. The inclusive token “ax” will include those algorithms.

Table 168 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_ASYM\_SCHEME Type <IO>

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.am | key exchange methods |
| TPM\_ALG\_!ALG.ax | all signing including anonymous |
| TPM\_ALG\_!ALG.ae | encrypting schemes |
| +TPM\_ALG\_NULL |  |
| #TPM\_RC\_VALUE |  |

#### TPMU\_ASYM\_SCHEME

This union of all asymmetric schemes is used in each of the asymmetric scheme structures. The actual scheme structure is defined by the interface type used for the selector (TPMI\_ALG\_ASYM\_SCHEME).

EXAMPLE The TPMT\_RSA\_SCHEME structure uses the TPMU\_ASYM\_SCHEME union but the selector type is TPMI\_ALG\_RSA\_SCHEME. This means that the only elements of the union that can be selected for the TPMT\_RSA\_SCHEME are those that are in TPMI\_RSA\_SCHEME.

Table 169 — Definition of TPMU\_ASYM\_SCHEME Union

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| !ALG.am | TPMS\_KEY\_SCHEME\_!ALG | TPM\_ALG\_!ALG |  |
| !ALG.ax | TPMS\_SIG\_SCHEME\_!ALG | TPM\_ALG\_!ALG | signing and anonymous signing |
| !ALG.ae | TPMS\_ENC\_SCHEME\_!ALG | TPM\_ALG\_!ALG | schemes with no hash |
| !ALG.HM | TPMS\_KDF\_SCHEME\_!ALG | TPM\_ALG\_!ALG | kdf schemes |
| anySig | TPMS\_SCHEME\_HASH |  |  |
| null |  | TPM\_ALG\_NULL | no scheme or default  This selects the NULL Signature. |

#### TPMT\_ASYM\_SCHEME

This structure is defined to allow overlay of all of the schemes for any asymmetric object. This structure is not sent on the interface. It is defined so that common functions may operate on any similar scheme structure.

EXAMPLE Since many schemes have a hash algorithm as their defining parameter, a common function can use the digest selector to select the hash of the scheme without a need to cast or use a large switch statement.

Table 170 — Definition of TPMT\_ASYM\_SCHEME Structure <>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| scheme | +TPMI\_ALG\_ASYM\_SCHEME | scheme selector |
| [scheme]details | TPMU\_ASYM\_SCHEME | scheme parameters |

### RSA

#### TPMI\_ALG\_RSA\_SCHEME

The list of values that may appear in the scheme parameter of a TPMS\_RSA\_PARMS structure.

Table 171 — Definition of (TPM\_ALG\_ID) {RSA} TPMI\_ALG\_RSA\_SCHEME Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.ae.ax | encrypting and signing algorithms |
| +TPM\_ALG\_NULL |  |
| #TPM\_RC\_VALUE |  |

#### TPMT\_RSA\_SCHEME

Table 172 — Definition of {RSA} TPMT\_RSA\_SCHEME Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| scheme | +TPMI\_ALG\_RSA\_SCHEME | scheme selector |
| [scheme]details | TPMU\_ASYM\_SCHEME | scheme parameters |

#### TPMI\_ALG\_RSA\_DECRYPT

The list of values that are allowed in a decryption scheme selection as used in TPM2\_RSA\_Encrypt() and TPM2\_RSA\_Decrypt().

Table 173 — Definition of (TPM\_ALG\_ID) {RSA} TPMI\_ALG\_RSA\_DECRYPT Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.ae | all RSA encryption algorithms |
| +TPM\_ALG\_NULL |  |
| #TPM\_RC\_VALUE |  |

#### TPMT\_RSA\_DECRYPT

Table 174 — Definition of {RSA} TPMT\_RSA\_DECRYPT Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| scheme | +TPMI\_ALG\_RSA\_DECRYPT | scheme selector |
| [scheme]details | TPMU\_ASYM\_SCHEME | scheme parameters |

#### TPM2B\_PUBLIC\_KEY\_RSA

This sized buffer holds the largest RSA public key supported by the TPM.

NOTE The reference implementation only supports key sizes of 1,024 and 2,048 bits.

Table 175 — Definition of {RSA} TPM2B\_PUBLIC\_KEY\_RSA Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size of the buffer  The value of zero is only valid for create. |
| buffer[size] {: MAX\_RSA\_KEY\_BYTES} | BYTE | Value |

#### TPMI\_RSA\_KEY\_BITS

This holds the value that is the maximum size allowed for an RSA key.

NOTE 1 An implementation is allowed to provide limited support for smaller RSA key sizes. That is, a TPM may be able to accept a smaller RSA key size in TPM2\_LoadExternal() when only the public area is loaded but not accept that smaller key size in any command that loads both the public and private portions of an RSA key. This would allow the TPM to validate signatures using the smaller key but would prevent the TPM from using the smaller key size for any other purpose.

NOTE 2 The definition for RSA\_KEY\_SIZES\_BITS used in the reference implementation is found in TPM 2.0 Part 4, Implementation.h

Table 176 — Definition of {RSA} (TPM\_KEY\_BITS) TPMI\_RSA\_KEY\_BITS Type

|  |  |
| --- | --- |
| Parameter | Description |
| $RSA\_KEY\_SIZES\_BITS | the number of bits in the supported key |
| #TPM\_RC\_VALUE | error when key size is not supported |

#### TPM2B\_PRIVATE\_KEY\_RSA

This sized buffer holds the largest RSA prime number supported by the TPM.

NOTE 1 All primes are required to have exactly half the number of significant bits as the public modulus, and the square of each prime is required to have the same number of significant bits as the public modulus.

NOTE 2 RSA\_PRIVATE\_SIZE is a vendor specific value that can be (MAX\_RSA\_KEY\_BYTES / 2) or ((MAX\_RSA\_KEY\_BYTES \* 5) ./ 2. The larger size would only apply to keys that have *fixedTPM* parents. The larger size was added in revision 01.53.

Table 177 — Definition of {RSA} TPM2B\_PRIVATE\_KEY\_RSA Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 |  |
| buffer[size]{:RSA\_PRIVATE\_SIZE } | BYTE |  |

### ECC

#### TPM2B\_ECC\_PARAMETER

This sized buffer holds the largest ECC parameter (coordinate) supported by the TPM.

Table 178 — Definition of TPM2B\_ECC\_PARAMETER Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size of *buffer* |
| buffer[size] {:MAX\_ECC\_KEY\_BYTES} | BYTE | the parameter data |

#### TPMS\_ECC\_POINT

This structure holds two ECC coordinates that, together, make up an ECC point.

Table 179 — Definition of {ECC} TPMS\_ECC\_POINT Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| x | TPM2B\_ECC\_PARAMETER | X coordinate |
| y | TPM2B\_ECC\_PARAMETER | Y coordinate |

#### TPM2B\_ECC\_POINT

This structure is defined to allow a point to be a single sized parameter so that it may be encrypted.

NOTE If the point is to be omitted, the X and Y coordinates need to be individually set to Empty Buffers. The minimum value for size will be four. It is checked indirectly by unmarshaling of the TPMS\_ECC\_POINT. If the type of *point* were BYTE, then *size* could have been zero. However, this would complicate the process of marshaling the structure.

Table 180 — Definition of {ECC} TPM2B\_ECC\_POINT Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size= | UINT16 | size of the remainder of this structure |
| point | TPMS\_ECC\_POINT | coordinates |
| #TPM\_RC\_SIZE |  | error returned if the unmarshaled size of *point* is not exactly equal to *size* |

#### TPMI\_ALG\_ECC\_SCHEME

Table 181 — Definition of (TPM\_ALG\_ID) {ECC} TPMI\_ALG\_ECC\_SCHEME Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.ax | the ecc signing schemes |
| TPM\_ALG\_!ALG.am | key exchange methods |
| +TPM\_ALG\_NULL |  |
| #TPM\_RC\_SCHEME |  |

#### TPMI\_ECC\_CURVE

This type enumerates the ECC curves implemented by the TPM.

Table 182 — Definition of {ECC} (TPM\_ECC\_CURVE) TPMI\_ECC\_CURVE Type

|  |  |
| --- | --- |
| Parameter | Description |
| $ECC\_CURVES | the list of implemented curves |
| #TPM\_RC\_CURVE | error when curve is not supported |

#### TPMT\_ECC\_SCHEME

Table 183 — Definition of (TPMT\_SIG\_SCHEME) {ECC} TPMT\_ECC\_SCHEME Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| scheme | +TPMI\_ALG\_ECC\_SCHEME | scheme selector |
| [scheme]details | TPMU\_ASYM\_SCHEME | scheme parameters |

#### TPMS\_ALGORITHM\_DETAIL\_ECC

This structure is used to report on the curve parameters of an ECC curve. It is returned by TPM2\_ECC\_Parameters().

Table 184 — Definition of {ECC} TPMS\_ALGORITHM\_DETAIL\_ECC Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| curveID | TPM\_ECC\_CURVE | identifier for the curve |
| keySize | UINT16 | Size in bits of the key |
| kdf | TPMT\_KDF\_SCHEME+ | if not TPM\_ALG\_NULL, the required KDF and hash algorithm used in secret sharing operations |
| sign | TPMT\_ECC\_SCHEME+ | If not TPM\_ALG\_NULL, this is the mandatory signature scheme that is required to be used with this curve. |
| p | TPM2B\_ECC\_PARAMETER | *Fp* (the modulus) |
| a | TPM2B\_ECC\_PARAMETER | coefficient of the linear term in the curve equation |
| b | TPM2B\_ECC\_PARAMETER | constant term for curve equation |
| gX | TPM2B\_ECC\_PARAMETER | x coordinate of base point G |
| gY | TPM2B\_ECC\_PARAMETER | y coordinate of base point G |
| n | TPM2B\_ECC\_PARAMETER | order of G |
| h | TPM2B\_ECC\_PARAMETER | cofactor (a size of zero indicates a cofactor of 1) |

## Signatures

### TPMS\_SIGNATURE\_RSA

Table 185 — Definition of {RSA} TPMS\_SIGNATURE\_RSA Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| hash | TPMI\_ALG\_HASH | the hash algorithm used to digest the message  TPM\_ALG\_NULL is not allowed. |
| sig | TPM2B\_PUBLIC\_KEY\_RSA | The signature is the size of a public key. |

Table 186 — Definition of Types for {RSA} Signature

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| TPMS\_SIGNATURE\_RSA | TPMS\_SIGNATURE\_!ALG.ax |  |

### TPMS\_SIGNATURE\_ECC

Table 187 — Definition of {ECC} TPMS\_SIGNATURE\_ECC Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| hash | TPMI\_ALG\_HASH | the hash algorithm used in the signature process  TPM\_ALG\_NULL is not allowed. |
| signatureR | TPM2B\_ECC\_PARAMETER |  |
| signatureS | TPM2B\_ECC\_PARAMETER |  |

Table 188 — Definition of Types for {ECC} TPMS\_SIGNATURE\_ECC

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| TPMS\_SIGNATURE\_ECC | TPMS\_SIGNATURE\_!ALG.ax |  |

### TPMU\_SIGNATURE

A TPMU\_SIGNATURE\_COMPOSITE is a union of the various signatures that are supported by a particular TPM implementation. The union allows substitution of any signature algorithm wherever a signature is required in a structure.

NOTE All TPM are required to support a hash algorithm and the HMAC algorithm.

When a symmetric algorithm is used for signing, the signing algorithm is assumed to be an HMAC based on the indicated hash algorithm. The HMAC key will either be referenced as part of the usage or will be implied by context.

Table 189 — Definition of TPMU\_SIGNATURE Union <IN/OUT>

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| !ALG.ax | TPMS\_SIGNATURE\_!ALG.ax | TPM\_ALG\_!ALG.ax | all asymmetric signatures |
| hmac | TPMT\_HA | TPM\_ALG\_HMAC | HMAC signature (required to be supported) |
| any | TPMS\_SCHEME\_HASH |  | used to access the hash |
| null |  | TPM\_ALG\_NULL | the NULL signature |

### TPMT\_SIGNATURE

Table 190 shows the basic algorithm-agile structure when a symmetric or asymmetric signature is indicated. The *sigAlg* parameter indicates the algorithm used for the signature. This structure is output from commands such as the attestation commands and TPM2\_Sign, and is an input to commands such as TPM2\_VerifySignature(), TPM2\_PolicySigned(), and TPM2\_FieldUpgradeStart().

Table 190 — Definition of TPMT\_SIGNATURE Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| sigAlg | +TPMI\_ALG\_SIG\_SCHEME | selector of the algorithm used to construct the signature |
| [sigAlg]signature | TPMU\_SIGNATURE | This shall be the actual signature information. |

## Key/Secret Exchange

### Introduction

The structures in 11.4 are used when a key or secret is being exchanged. The exchange may be in

* TPM2\_StartAuthSession() where the secret is injected for salting the session,
* TPM2\_Duplicate(), TPM2\_Import, or TPM2\_Rewrap() where the secret is the symmetric encryption key for the outer wrapper of a duplication blob, or
* TPM2\_ActivateIdentity() or TPM2\_CreateIdentity() where the secret is the symmetric encryption key for the credential blob.

Particulars are described in TPM 2.0 Part 1.

### TPMU\_ENCRYPTED\_SECRET

This structure is used to hold either an ephemeral public point for ECDH, an OAEP-encrypted block for RSA, or a symmetrically encrypted value. This structure is defined for the limited purpose of determining the size of a TPM2B\_ENCRYPTED\_SECRET.

The symmetrically encrypted value may use either CFB or XOR encryption.

NOTE Table 191 is illustrative. It would be modified depending on the algorithms supported in the TPM.

Table 191 — Definition of TPMU\_ENCRYPTED\_SECRET Union

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| ecc[sizeof(TPMS\_ECC\_POINT)] | BYTE | TPM\_ALG\_ECC |  |
| rsa[MAX\_RSA\_KEY\_BYTES] | BYTE | TPM\_ALG\_RSA |  |
| symmetric[sizeof(TPM2B\_DIGEST)] | BYTE | TPM\_ALG\_SYMCIPHER |  |
| keyedHash[sizeof(TPM2B\_DIGEST)] | BYTE | TPM\_ALG\_KEYEDHASH | Any symmetrically encrypted secret value will be limited to be no larger than a digest. |

### TPM2B\_ENCRYPTED\_SECRET

Table 192 — Definition of TPM2B\_ENCRYPTED\_SECRET Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size of the secret value |
| secret[size] {:sizeof(TPMU\_ENCRYPTED\_SECRET)} | BYTE | secret |

# Key/Object Complex

## Introduction

An object description requires a TPM2B\_PUBLIC structure and may require a TPMT\_SENSITIVE structure. When the structure is stored off the TPM, the TPMT\_SENSITIVE structure is encrypted within a TPM2B\_PRIVATE structure.

When the object requires two components for its description, those components are loaded as separate parameters in the TPM2\_Load() command. When the TPM creates an object that requires both components, the TPM will return them as separate parameters from the TPM2\_Create() operation.

The TPM may produce multiple different TPM2B\_PRIVATE structures for a single TPM2B\_PUBLIC structure. Creation of a modified TPM2B\_PRIVATE structure requires that the full structure be loaded with the TPM2\_Load() command, modification of the TPMT\_SENSITIVE data, and output of a new TPM2B\_PRIVATE structure.

## Public Area Structures

### Description

Clause 12.2 defines the TPM2B\_PUBLIC structure and the higher-level substructure that may be contained in a TPM2B\_PUBLIC. The higher-level structures that are currently defined for inclusion in a TPM2B\_PUBLIC are the

* structures for asymmetric keys,
* structures for symmetric keys, and
* structures for sealed data.

### TPMI\_ALG\_PUBLIC

Table 193 — Definition of (TPM\_ALG\_ID) TPMI\_ALG\_PUBLIC Type

| Values | Comments |
| --- | --- |
| TPM\_ALG\_!ALG.o | All object types |
| #TPM\_RC\_TYPE | response code when a public type is not supported |

### Type-Specific Parameters

#### Description

The public area contains two fields (*parameters* and *unique*) that vary by object type. The *parameters* field varies according to the *type* of the object but the contents may be the same across multiple instances of a particular *type*. The unique field format also varies according to the type of the object and will also be unique for each instance.

For a symmetric key (*type* == TPM\_ALG\_SYMCIPHER), HMAC key (*type* == TPM\_ALG\_KEYEDHASH) or data object (also, *type* == TPM\_ALG\_KEYEDHASH), the contents of *unique* shall be computed from components of the sensitive area of the object as follows:

unique ≔ **H**nameAlg(seedValue || sensitive) 1

where

**H**nameAlg() the hash algorithm used to compute the Name of the object

seedValue the digest-sized obfuscation value in the sensitive area of a symmetric key or symmetric data object found in a TPMT\_SENSITIVE.seedValue.buffer

sensitive the secret key/data of the object in the TPMT\_SENSITIVE.sensitive.any.buffer

#### TPMU\_PUBLIC\_ID

This is the union of all values allowed in in the *unique* field of a TPMT\_PUBLIC.

NOTE The derive member cannot be unmarshaled in a TPMU\_PUBLIC\_ID. It is placed in this structure so that the maximum size of a TPM2B\_TEMPLATE will be computed correctly.

Table 194 — Definition of TPMU\_PUBLIC\_ID Union <IN/OUT>

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| keyedHash | TPM2B\_DIGEST | TPM\_ALG\_KEYEDHASH |  |
| sym | TPM2B\_DIGEST | TPM\_ALG\_SYMCIPHER |  |
| rsa | TPM2B\_PUBLIC\_KEY\_RSA | TPM\_ALG\_RSA |  |
| ecc | TPMS\_ECC\_POINT | TPM\_ALG\_ECC |  |
| derive | TPMS\_DERIVE |  | only allowed for TPM2\_CreateLoaded when *parentHandle*  is a Derivation Parent. |

#### TPMS\_KEYEDHASH\_PARMS

This structure describes the parameters that would appear in the public area of a KEYEDHASH object.

NOTE Although the names are the same, the types of the structures are not the same as for asymmetric parameter lists.

Table 195 — Definition of TPMS\_KEYEDHASH\_PARMS Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| scheme | TPMT\_KEYEDHASH\_SCHEME+ | Indicates the signing method used for a *keyedHash* signing object. This field also determines the size of the data field for a data object created with TPM2\_Create() or TPM2\_CreatePrimary(). |

#### TPMS\_ASYM\_PARMS

This structure contains the common public area parameters for an asymmetric key. The first two parameters of the parameter definition structures of an asymmetric key shall have the same two first components.

NOTE The sign parameter may have a different type in order to allow different schemes to be selected for each asymmetric type but the first parameter of each scheme definition shall be a TPM\_ALG\_ID for a valid signing scheme.

Table 196 — Definition of TPMS\_ASYM\_PARMS Structure <>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| symmetric | TPMT\_SYM\_DEF\_OBJECT+ | the companion symmetric algorithm for a restricted decryption key and shall be set to a supported symmetric algorithm  This field is optional for keys that are not decryption keys and shall be set to TPM\_ALG\_NULL if not used. |
| scheme | TPMT\_ASYM\_SCHEME+ | for a key with the *sign* attribute SET, a valid signing scheme for the key type  for a key with the *decrypt* attribute SET, a valid key exchange protocol  for a key with sign and decrypt attributes, shall be TPM\_ALG\_NULL |

#### TPMS\_RSA\_PARMS

A TPM compatible with this specification and supporting RSA shall support two primes and an *exponent* of zero. An exponent of zero indicates that the exponent is the default of 216 + 1. Support for other values is optional. Use of other exponents in duplicated keys is not recommended because the resulting keys would not be interoperable with other TPMs.

NOTE Implementations are not required to check that *exponent* is the default exponent. They may fail to load the key if *exponent* is not zero. The reference implementation allows the values listed in the table.

Table 197 — Definition of {RSA} TPMS\_RSA\_PARMS Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| symmetric | TPMT\_SYM\_DEF\_OBJECT+ | for a restricted decryption key, shall be set to a supported symmetric algorithm, key size, and mode.  if the key is not a restricted decryption key, this field shall be set to TPM\_ALG\_NULL. |
| scheme | TPMT\_RSA\_SCHEME+ | scheme.scheme shall be:  for an unrestricted signing key, either TPM\_ALG\_RSAPSS TPM\_ALG\_RSASSA or TPM\_ALG\_NULL  for a restricted signing key, either TPM\_ALG\_RSAPSS or TPM\_ALG\_RSASSA  for an unrestricted decryption key, TPM\_ALG\_RSAES, TPM\_ALG\_OAEP, or TPM\_ALG\_NULL unless the object also has the *sign* attribute  for a restricted decryption key, TPM\_ALG\_NULL  NOTE When both sign and decrypt are SET, restricted shall be CLEAR and scheme shall be TPM\_ALG\_NULL. |
| keyBits | TPMI\_RSA\_KEY\_BITS | number of bits in the public modulus |
| exponent | UINT32 | the public exponent  A prime number greater than 2. |

#### TPMS\_ECC\_PARMS

This structure contains the parameters for prime modulus ECC.

Table 198 — Definition of {ECC} TPMS\_ECC\_PARMS Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| symmetric | TPMT\_SYM\_DEF\_OBJECT+ | for a restricted decryption key, shall be set to a supported symmetric algorithm, key size. and mode.  if the key is not a restricted decryption key, this field shall be set to TPM\_ALG\_NULL. |
| scheme | TPMT\_ECC\_SCHEME+ | If the *sign* attribute of the key is SET, then this shall be a valid signing scheme.  NOTE If the *sign* parameter in *curveID* indicates a mandatory scheme, then this field shall have the same value.  If the *decrypt* attribute of the key is SET, then this shall be a valid key exchange scheme or TPM\_ALG\_NULL.  If the key is a Storage Key, then this field shall be TPM\_ALG\_NULL. |
| curveID | TPMI\_ECC\_CURVE | ECC curve ID |
| kdf | TPMT\_KDF\_SCHEME+ | an optional key derivation scheme for generating a symmetric key from a Z value  If the *kdf*  parameter associated with *curveID* is not TPM\_ALG\_NULL then this is required to be NULL.  NOTE There are currently no commands where this parameter has effect and, in the reference code, this field needs to be set to TPM\_ALG\_NULL. |

#### TPMU\_PUBLIC\_PARMS

Table 199 defines the possible parameter definition structures that may be contained in the public portion of a key. If the Object can be a parent, the first field must be a TPMT\_SYM\_DEF\_OBJECT. See 11.1.7.

Table 199 — Definition of TPMU\_PUBLIC\_PARMS Union <IN/OUT>

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description(1) |
| keyedHashDetail | TPMS\_KEYEDHASH\_PARMS | TPM\_ALG\_KEYEDHASH | sign | decrypt | neither(2) |
| symDetail | TPMS\_SYMCIPHER\_PARMS | TPM\_ALG\_SYMCIPHER | sign | decrypt | neither(2) |
| rsaDetail | TPMS\_RSA\_PARMS | TPM\_ALG\_RSA | decrypt + sign(2) |
| eccDetail | TPMS\_ECC\_PARMS | TPM\_ALG\_ECC | decrypt + sign(2) |
| asymDetail | TPMS\_ASYM\_PARMS |  | common scheme structure for RSA and ECC keys |
| NOTES  1) Description column indicates which of TPMA\_OBJECT.*decrypt* or TPMA\_OBJECT.*sign* may be set.  2) “+” indicates that both may be set but one shall be set. “|” indicates the optional settings. | | | |

#### TPMT\_PUBLIC\_PARMS

This structure is used in TPM2\_TestParms() to validate that a set of algorithm parameters is supported by the TPM.

Table 200 — Definition of TPMT\_PUBLIC\_PARMS Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| type | TPMI\_ALG\_PUBLIC | the algorithm to be tested |
| [type]parameters | TPMU\_PUBLIC\_PARMS | the algorithm details |

### TPMT\_PUBLIC

Table 201 defines the public area structure. The Name of the object is *nameAlg* concatenated with the digest of this structure using *nameAlg*.

Table 201 — Definition of TPMT\_PUBLIC Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| type | TPMI\_ALG\_PUBLIC | “algorithm” associated with this object |
| nameAlg | +TPMI\_ALG\_HASH | algorithm used for computing the Name of the object  NOTE The "+" indicates that the instance of a TPMT\_PUBLIC may have a "+" to indicate that the *nameAlg* may be TPM\_ALG\_NULL. |
| objectAttributes | TPMA\_OBJECT | attributes that, along with *type*, determine the manipulations of this object |
| authPolicy | TPM2B\_DIGEST | optional policy for using this key  The policy is computed using the *nameAlg* of the object.  NOTE Shall be the Empty Policy if no authorization policy is present. |
| [type]parameters | TPMU\_PUBLIC\_PARMS | the algorithm or structure details |
| [type]unique | TPMU\_PUBLIC\_ID | the unique identifier of the structure  For an asymmetric key, this would be the public key. |

### TPM2B\_PUBLIC

This sized buffer is used to embed a TPMT\_PUBLIC in a load command and in any response that returns a public area.

Table 202 — Definition of TPM2B\_PUBLIC Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size= | UINT16 | size of publicArea  NOTE The “=” will force the TPM to try to unmarshal a TPMT\_PUBLIC and check that the unmarshaled size matches the value of *size.* If all the required fields of a TPMT\_PUBLIC are not present, the TPM will return an error (generally TPM\_RC\_SIZE) when attempting to unmarshal the TPMT\_PUBLIC. |
| publicArea | +TPMT\_PUBLIC | the public area  NOTE The “+” indicates that the caller may specify that use of TPM\_ALG\_NULL is allowed for *nameAlg.* |

### TPM2B\_TEMPLATE

This sized buffer is used to embed a TPMT\_TEMPLATE for TPM2\_CreateLoaded().

Unmarshaling of this structure is fairly complex due to requirements for backwards compatibility. Unlike a TPM2B\_PUBLIC, this structure is unmarshaled as an array of bytes that is passed to the action code. The action code will then unmarshal the embedded structure.

If the parent is not a derivation parent, this structure is unmarshaled normally. If the parent is a derivation parent, *unique* is unmarshaled as a TPMS\_DERIVE structure (*label* and *context*). See 12.2.3.2.

Table 203 — Definition of TPM2B\_TEMPLATE Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size of publicArea |
| buffer[size]{:sizeof(TPMT\_PUBLIC)} | BYTE | the public area |

## Private Area Structures

### Introduction

The structures in 12.2.6 define the contents and construction of the private portion of a TPM object. A TPM2B\_PRIVATE along with a TPM2B\_PUBLIC are needed to describe a TPM object.

A TPM2B\_PRIVATE area may be encrypted by different symmetric algorithms or, in some cases, not encrypted at all.

### Sensitive Data Structures

#### Introduction

The structures in 12.3.2 define the presumptive internal representations of the sensitive areas of the various entities. A TPM may store the sensitive information in any desired format but when constructing a TPM\_PRIVATE, the formats in 12.3.2 shall be used.

#### TPM2B\_PRIVATE\_VENDOR\_SPECIFIC

This structure is defined for coding purposes. For IO to the TPM, the sensitive portion of the key will be in a canonical form. For an RSA key, this will be one of the prime factors of the public modulus. After loading, it is typical that other values will be computed so that computations using the private key will not need to start with just one prime factor. This structure can be used to store the results of such vendor-specific calculations.

The value for PRIVATE\_VENDOR\_SPECIFIC\_BYTES is determined by the vendor.

Table 204 — Definition of TPM2B\_PRIVATE\_VENDOR\_SPECIFIC Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 |  |
| buffer[size]{:PRIVATE\_VENDOR\_SPECIFIC\_BYTES} | BYTE |  |

#### TPMU\_SENSITIVE\_COMPOSITE

Table 205 — Definition of TPMU\_SENSITIVE\_COMPOSITE Union <IN/OUT>

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | Type | Selector | Description |
| rsa | TPM2B\_PRIVATE\_KEY\_RSA | TPM\_ALG\_RSA | a prime factor of the public key |
| ecc | TPM2B\_ECC\_PARAMETER | TPM\_ALG\_ECC | the integer private key |
| bits | TPM2B\_SENSITIVE\_DATA | TPM\_ALG\_KEYEDHASH | the private data |
| sym | TPM2B\_SYM\_KEY | TPM\_ALG\_SYMCIPHER | the symmetric key |
| any | TPM2B\_PRIVATE\_VENDOR\_SPECIFIC |  | vendor-specific size for key storage |

#### TPMT\_SENSITIVE

*authValue* shall not be larger than the size of the digest produced by the *nameAlg* of the object. *seedValue* shall be the size of the digest produced by the *nameAlg* of the object.

Table 206 — Definition of TPMT\_SENSITIVE Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| sensitiveType | TPMI\_ALG\_PUBLIC | identifier for the sensitive area  This shall be the same as the *type* parameter of the associated public area. |
| authValue | TPM2B\_AUTH | user authorization data  The authValue may be a zero-length string. |
| seedValue | TPM2B\_DIGEST | for a parent object, the optional protection seed; for other objects, the obfuscation value |
| [sensitiveType]sensitive | TPMU\_SENSITIVE\_COMPOSITE | the type-specific private data |

### TPM2B\_SENSITIVE

The TPM2B\_SENSITIVE structure is used as a parameter in TPM2\_LoadExternal(). It is an unencrypted sensitive area but it may be encrypted using parameter encryption.

NOTE 1 When this structure is unmarshaled, the *sensitiveType* determines what type of value is unmarshaled. Each value of *sensitiveType* is associated with a TPM2B. It is the maximum size for each of the TPM2B values that will determine if the unmarshal operation is successful. Since there is no selector for the *any* or *vendor* options for the union, the maximum input and output sizes for a TPM2B\_SENSITIVE are not affected by the sizes of those parameters.

NOTE 2 The unmarshaling function validates that *size* equals the size of the value that is unmarshaled.

Table 207 — Definition of TPM2B\_SENSITIVE Structure <IN/OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size of the *private* structure |
| sensitiveArea | TPMT\_SENSITIVE | an unencrypted sensitive area |

### Encryption

A TPMS\_SENSITIVE is the input to the encryption process. All TPMS\_ENCRYPT structures are CFB-encrypted using a key and Initialization Vector (IV) that are derived from a seed value.

The method of generating the key and IV is described in “Protected Storage” subclause “Symmetric Encryption.” in TPM 2.0 Part 1.

### Integrity

The integrity computation is used to ensure that a protected object is not modified when stored in memory outside of the TPM.

The method of protecting the integrity of the sensitive area is described in “Protected Storage” subclause “Integrity” in TPM 2.0 Part 1.

### \_PRIVATE

This structure is defined to size the contents of a TPM2B\_PRIVATE. This structure is not directly marshaled or unmarshaled.

For TPM2\_Duplicate() and TPM2\_Import(), the TPM2B\_PRIVATE may contain multiply encrypted data and two integrity values. In some cases, the sensitive data is not encrypted and the integrity value is not present.

For TPM2\_Load() and TPM2\_Create(), *integrityInner* is always present.

If *integrityInner* is present, it and *sensitive* are encrypted as a single block.

When an integrity value is not needed, it is not present and it is not represented by an Empty Buffer.

Table 208 — Definition of \_PRIVATE Structure <>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| integrityOuter | TPM2B\_DIGEST |  |
| integrityInner | TPM2B\_DIGEST | could also be a TPM2B\_IV |
| sensitive | TPM2B\_SENSITIVE | the sensitive area |

### TPM2B\_PRIVATE

The TPM2B\_PRIVATE structure is used as a parameter in multiple commands that create, load, and modify the sensitive area of an object.

When the TPM returns a TPM2B\_PRIVATE structure, the TPM pads the TPM2B\_AUTH to its maximum size.

Table 209 — Definition of TPM2B\_PRIVATE Structure <IN/OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size of the *private* structure |
| buffer[size] {:sizeof(\_PRIVATE)} | BYTE | an encrypted private area |

## Identity Object

### Description

An identity object is used to convey credential protection value (CV) to a TPM that can load the object associated with the object. The CV is encrypted to a storage key on the target TPM, and if the credential integrity checks and the proper object is loaded in the TPM, then the TPM will return the CV.

### TPMS\_ID\_OBJECT

This structure is used for sizing the TPM2B\_ID\_OBJECT.

Table 210 — Definition of TPMS\_ID\_OBJECT Structure <>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| integrityHMAC | TPM2B\_DIGEST | HMAC using the nameAlg of the storage key on the target TPM |
| encIdentity | TPM2B\_DIGEST | credential protector information returned if name matches the referenced object  All of the *encIdentity* is encrypted, including the size field.  NOTE The TPM is not required to check that the size is not larger than the digest of the *nameAlg*. However, if the size is larger, the ID object may not be usable on a TPM that has no digest larger than produced by *nameAlg*. |

### TPM2B\_ID\_OBJECT

This structure is an output from TPM2\_MakeCredential() and is an input to TPM2\_ActivateCredential().

Table 211 — Definition of TPM2B\_ID\_OBJECT Structure <IN/OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 | size of the *credential* structure |
| credential[size]{:sizeof(TPMS\_ID\_OBJECT)} | BYTE | an encrypted credential area |

# NV Storage Structures

## TPM\_NV\_INDEX

A TPM\_NV\_INDEX is used to reference a defined location in NV memory. The format of the Index is changed from TPM 1.2 in order to include the Index in the reserved handle space. Handles in this range use the digest of the public area of the Index as the Name of the entity in authorization computations

The 32-bit TPM 1.2 NV Index format is shown in Figure 4. In order to allow the Index to fit into the 24 bits available in the reserved handle space, the Index value format is changed as shown in Figure 5.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3  1 | 3  0 | 2  9 | 2  8 | 2  7 | 2  6 | 2  5 | 2  4 | 2  3 |  |  |  |  |  |  | 1  6 | 1  5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0  0 |
| T | P | U | D | reserved | | | | Purview | | | | | | | | Index | | | | | | | | | | | | | | | |

Figure 4 — TPM 1.2 TPM\_NV\_INDEX

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3  1 |  |  |  |  |  |  | 2  4 | 2  3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | 0  0 |
| TPM\_HT\_NV\_INDEX | | | | | | | | Index | | | | | | | | | | | | | | | | | | | | | |

Figure 5 — TPM 2.0 TPM\_NV\_INDEX

NOTE This TPM\_NV\_INDEX format does not retain the Purview field and the D bit is not a part of an Index handle as in TPM 1.2. The TPMA\_NV\_PLATFORMCREATE attribute is a property of an Index that provides functionality similar to the D bit.

A valid Index handle will have an MSO of TPM\_HT\_NV\_INDEX.

NOTE This structure is not used. It is defined here to indicate how the fields of the handle are assigned. The exemplary unmarshaling code unmarshals a TPM\_HANDLE and validates that it is in the range for a TPM\_NV\_INDEX.

Table 212 — Definition of (UINT32) TPM\_NV\_INDEX Bits <>

|  |  |  |
| --- | --- | --- |
| Bit | Name | Definition |
| 23:00 | index | The Index of the NV location |
| 31:24 | RH\_NV | constant value of TPM\_HT\_NV\_INDEX indicating the NV Index range |

Some prior versions of this specification contained a table here (Options for space Field of TPM\_NV\_INDEX) that assigned subsets of the index field to different entities. Since this assignment was a convention and not an architectural element of the TPM, the table was removed and the information is now contained in a registry document that is maintained by the TCG.

## TPM\_NT

This table lists the values of the TPM\_NT field of a TPMA\_NV. See Table 215 for usage.

Table 213 — Definition of TPM\_NT Constants

| Name | Value | Description |
| --- | --- | --- |
| TPM\_NT\_ORDINARY | 0x0 | Ordinary – contains data that is opaque to the TPM that can only be modified using TPM2\_NV\_Write(). |
| TPM\_NT\_COUNTER | 0x1 | Counter – contains an 8-octet value that is to be used as a counter and can only be modified with TPM2\_NV\_Increment() |
| TPM\_NT\_BITS | 0x2 | Bit Field – contains an 8-octet value to be used as a bit field and can only be modified with TPM2\_NV\_SetBits(). |
| TPM\_NT\_EXTEND | 0x4 | Extend – contains a digest-sized value used like a PCR. The Index can only be modified using TPM2\_NV\_Extend(). The extend will use the nameAlg of the Index. |
| TPM\_NT\_PIN\_FAIL | 0x8 | PIN Fail - contains *pinCount* that increments on a PIN authorization failure and a *pinLimit* |
| TPM\_NT\_PIN\_PASS | 0x9 | PIN Pass - contains *pinCount* that increments on a PIN authorization success and a *pinLimit* |

All other TPM\_NT values are reserved and TPM2\_NV\_DefineSpace() returns TPM\_RC\_ATTRIBUTES.

NOTE 1 These values are compatible with previous versions of this specification, which used a bit map for this field.

NOTE 2 This field described by Table 213 is 4 bits.

## TPMS\_NV\_PIN\_COUNTER\_PARAMETERS

This is the data that can be written to and read from a TPM\_NT\_PIN\_PASS or TPM\_NT\_PIN\_FAIL non-volatile index. *pinCount* is the most significant octets. *pinLimit* is the least significant octets.

Table 214 — Definition of TPMS\_NV\_PIN\_COUNTER\_PARAMETERS Structure

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| pinCount | UINT32 | This counter shows the current number of successful authValue authorization attempts to access a TPM\_NT\_PIN\_PASS index or the current number of unsuccessful authValue authorization attempts to access a TPM\_NT\_PIN\_FAIL index. |
| pinLimit | UINT32 | This threshold is the value of *pinCount* at which the authValue authorization of the host TPM\_NT\_PIN\_PASS or TPM\_NT\_PIN\_FAIL index is locked out. |

## TPMA\_NV (NV Index Attributes)

This structure allows the TPM to keep track of the data and permissions to manipulate an NV Index.

The platform controls (TPMA\_NV\_PPWRITE and TPMA\_NV\_PPREAD) and owner controls (TPMA\_NV\_OWNERWRITE and TPMA\_NV\_OWNERREAD) give the platform and owner access to NV Indexes using Platform Authorization or Owner Authorization rather than the *authValue* or *authPolicy* of the Index.

If access to an NV Index is to be restricted based on PCR, then an appropriate *authPolicy* shall be provided.

NOTE *platformAuth* or *ownerAuth* can be provided in any type of authorization session or as a password.

If TPMA\_NV\_AUTHREAD is SET, then the Index may be read if the Index *authValue* is provided. If TPMA\_NV\_POLICYREAD is SET, then the Index may be read if the Index *authPolicy* is satisfied.

At least one of TPMA\_NV\_PPREAD, TPMA\_NV\_OWNERREAD, TPMA\_NV\_AUTHREAD, or TPMA\_NV\_POLICYREAD shall be SET.

If TPMA\_NV\_AUTHWRITE is SET, then the Index may be written if the Index *authValue* is provided. If TPMA\_NV\_POLICYWRITE is SET, then the Index may be written if the Index *authPolicy* is satisfied.

At least one of TPMA\_NV\_PPWRITE, TPMA\_NV\_OWNERWRITE TPMA\_NV\_AUTHWRITE, or TPMA\_NV\_POLICYWRITE shall be SET.

If TPMA\_NV\_WRITELOCKED is SET, then the Index may not be written. If TPMA\_NV\_WRITEDEFINE is SET, TPMA\_NV\_WRITELOCKED may not be CLEAR except by deleting and redefining the Index. If TPMA\_NV\_WRITEDEFINE is CLEAR, then TPMA\_NV\_WRITELOCKED will be CLEAR on the next TPM2\_Startup(TPM\_SU\_CLEAR).

NOTE If TPMA\_NV\_WRITELOCKED is SET, but TPMA\_NV\_WRITTEN is CLEAR, then TPMA\_NV\_WRITELOCKED is CLEAR by TPM Reset or TPM Restart. This action occurs even if the TPMA\_NV\_WRITEDEFINE attribute is SET. This action prevents an NV Index from being defined that can never be written, and permits a use case where an Index is defined, but the user wants to prohibit writes until after a reboot.

If TPMA\_NV\_READLOCKED is SET, then the Index may not be read. TPMA\_NV\_READLOCKED will be CLEAR on the next TPM2\_Startup(TPM\_SU\_CLEAR).

NOTE The TPM is expected to maintain indicators to indicate that the Index is temporarily locked. The state of these indicators is reported in the TPMA\_NV\_READLOCKED and TPMA\_NV\_WRITELOCKED attributes.

If the TPM\_NT is TPM\_NT\_EXTEND, then writes to the Index will cause an update of the Index using the extend operation with the *nameAlg* used to create the digest.

If TPM\_NT is TPM\_NT\_PIN\_FAIL, TPMA\_NV\_NO\_DA must be SET. This removes ambiguity over which Dictionary Attack defense protects a TPM\_NV\_PIN\_FAIL's *authValue*.

When the Index is created (TPM2\_NV\_DefineSpace()), TPMA\_NV\_WRITELOCKED, TPMA\_NV\_READLOCKED, and TPMA\_NV\_WRITTEN shall all be CLEAR in the parameter that defines the attributes of the created Index.

Table 215 — Definition of (UINT32) TPMA\_NV Bits

| Bit | Name | Description |
| --- | --- | --- |
| 0 | TPMA\_NV\_PPWRITE | **SET (1):** The Index data can be written if Platform Authorization is provided.  **CLEAR (0):** Writing of the Index data cannot be authorized with Platform Authorization. |
| 1 | TPMA\_NV\_OWNERWRITE | **SET (1):** The Index data can be written if Owner Authorization is provided.  **CLEAR (0):** Writing of the Index data cannot be authorized with Owner Authorization. |
| 2 | TPMA\_NV\_AUTHWRITE | **SET (1):** Authorizations to change the Index contents that require USER role may be provided with an HMAC session or password.  **CLEAR (0):** Authorizations to change the Index contents that require USER role may not be provided with an HMAC session or password. |
| 3 | TPMA\_NV\_POLICYWRITE | **SET (1):** Authorizations to change the Index contents that require USER role may be provided with a policy session.  **CLEAR (0):** Authorizations to change the Index contents that require USER role may not be provided with a policy session.  NOTE TPM2\_NV\_ChangeAuth() always requires that authorization be provided in a policy session. |
| 7:4 | TPM\_NT | The type of the index.  NOTE A TPM is not required to support all TPM\_NT values |
| 9:8 | Reserved | shall be zero  reserved for future use |
| 10 | TPMA\_NV\_POLICY\_DELETE | **SET (**1**):** Index may not be deleted unless the *authPolicy* is satisfied using TPM2\_NV\_UndefineSpaceSpecial().  **CLEAR (0):** Index may be deleted with proper platform or owner authorization using TPM2\_NV\_UndefineSpace().  NOTE An Index with this attribute and a policy that cannot be satisfied (e.g., an Empty Policy) cannot be deleted. |
| 11 | TPMA\_NV\_WRITELOCKED | **SET (1):** Index cannot be written.  **CLEAR (0):** Index can be written. |
| 12 | TPMA\_NV\_WRITEALL | **SET (1):** A partial write of the Index data is not allowed. The write size shall match the defined space size.  **CLEAR (0):** Partial writes are allowed. This setting is required if the .*dataSize* of the Index is larger than NV\_MAX\_BUFFER\_SIZE for the implementation. |
| 13 | TPMA\_NV\_WRITEDEFINE | **SET (1):** TPM2\_NV\_WriteLock() may be used to prevent further writes to this location.  **CLEAR (0):** TPM2\_NV\_WriteLock() does not block subsequent writes if TPMA\_NV\_WRITE\_STCLEAR is also CLEAR. |
| 14 | TPMA\_NV\_WRITE\_STCLEAR | **SET (1):** TPM2\_NV\_WriteLock() may be used to prevent further writes to this location until the next TPM Reset or TPM Restart.  **CLEAR (0):** TPM2\_NV\_WriteLock() does not block subsequent writes if TPMA\_NV\_WRITEDEFINE is also CLEAR. |
| 15 | TPMA\_NV\_GLOBALLOCK | **SET (1):** If TPM2\_NV\_GlobalWriteLock() is successful, TPMA\_NV\_WRITELOCKED is set.  **CLEAR (0):** TPM2\_NV\_GlobalWriteLock() has no effect on the writing of the data at this Index. |
| 16 | TPMA\_NV\_PPREAD | **SET (1):** The Index data can be read if Platform Authorization is provided.  **CLEAR (0):** Reading of the Index data cannot be authorized with Platform Authorization. |
| 17 | TPMA\_NV\_OWNERREAD | **SET (1):** The Index data can be read if Owner Authorization is provided.  **CLEAR (0):** Reading of the Index data cannot be authorized with Owner Authorization. |
| 18 | TPMA\_NV\_AUTHREAD | **SET (1):** The Index data may be read if the *authValue* is provided.  **CLEAR (0):** Reading of the Index data cannot be authorized with the Index *authValue*. |
| 19 | TPMA\_NV\_POLICYREAD | **SET (1):** The Index data may be read if the *authPolicy* is satisfied.  **CLEAR (0):** Reading of the Index data cannot be authorized with the Index *authPolicy*. |
| 24:20 | Reserved | shall be zero  reserved for future use |
| 25 | TPMA\_NV\_NO\_DA | **SET (1):** Authorization failures of the Index do not affect the DA logic and authorization of the Index is not blocked when the TPM is in Lockout mode.  **CLEAR (0):** Authorization failures of the Index will increment the authorization failure counter and authorizations of this Index are not allowed when the TPM is in Lockout mode. |
| 26 | TPMA\_NV\_ORDERLY | **SET (1):** NV Index state is only required to be saved when the TPM performs an orderly shutdown (TPM2\_Shutdown()).  **CLEAR (0):** NV Index state is required to be persistent after the command to update the Index completes successfully (that is, the NV update is synchronous with the update command). |
| 27 | TPMA\_NV\_CLEAR\_STCLEAR | **SET (1):** TPMA\_NV\_WRITTEN for the Index is CLEAR by TPM Reset or TPM Restart.  **CLEAR (0):** TPMA\_NV\_WRITTEN is not changed by TPM Restart.  NOTE This attribute may only be SET if TPM\_NT is not TPM\_NT\_COUNTER. |
| 28 | TPMA\_NV\_READLOCKED | **SET (1):** Reads of the Index are blocked until the next TPM Reset or TPM Restart.  **CLEAR (0):** Reads of the Index are allowed if proper authorization is provided. |
| 29 | TPMA\_NV\_WRITTEN | **SET (1):** Index has been written.  **CLEAR (0):** Index has not been written. |
| 30 | TPMA\_NV\_PLATFORMCREATE | **SET (1):** This Index may be undefined with Platform Authorization but not with Owner Authorization.  **CLEAR (0):** This Index may be undefined using Owner Authorization but not with Platform Authorization.  The TPM will validate that this attribute is SET when the Index is defined using Platform Authorization and will validate that this attribute is CLEAR when the Index is defined using Owner Authorization. |
| 31 | TPMA\_NV\_READ\_STCLEAR | **SET (1):** TPM2\_NV\_ReadLock() may be used to SET TPMA\_NV\_READLOCKED for this Index.  **CLEAR (0):** TPM2\_NV\_ReadLock() has no effect on this Index. |

## TPMS\_NV\_PUBLIC

This structure describes an NV Index.

Table 216 — Definition of TPMS\_NV\_PUBLIC Structure

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| nvIndex | TPMI\_RH\_NV\_INDEX | the handle of the data area |
| nameAlg | TPMI\_ALG\_HASH | hash algorithm used to compute the name of the Index and used for the *authPolicy.* For an extend index, the hash algorithm used for the extend. |
| attributes | TPMA\_NV | the Index attributes |
| authPolicy | TPM2B\_DIGEST | optional access policy for the Index  The policy is computed using the *nameAlg*  NOTE Shall be the Empty Policy if no authorization policy is present. |
| dataSize{:MAX\_NV\_INDEX\_SIZE} | UINT16 | the size of the data area  The maximum size is implementation-dependent. The minimum maximum size is platform-specific. |
| #TPM\_RC\_SIZE |  | response code returned when the requested size is too large for the implementation |

## TPM2B\_NV\_PUBLIC

This structure is used when a TPMS\_NV\_PUBLIC is sent on the TPM interface.

Table 217 — Definition of TPM2B\_NV\_PUBLIC Structure

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| size= | UINT16 | size of *nvPublic* |
| nvPublic | TPMS\_NV\_PUBLIC | the public area |

# Context Data

## Introduction

Clause 14 defines the contents of the TPM2\_ContextSave() response parameters and TPM2\_ContextLoad() command parameters.

If the parameters provided by the caller in TPM2\_ContextLoad() do not match the values returned by the TPM when the context was saved, the integrity check of the TPM2B\_CONTEXT will fail and the object or session will not be loaded.

## TPM2B\_CONTEXT\_SENSITIVE

This structure holds the object or session context data. When saved, the full structure is encrypted.

NOTE This is an informative table that is included in the specification only to allow calculation of the maximum size for TPM2B\_CONTEXT\_DATA.

Table 218 — Definition of TPM2B\_CONTEXT\_SENSITIVE Structure <IN/OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 |  |
| buffer[size]{:MAX\_CONTEXT\_SIZE} | BYTE | the sensitive data |

## TPMS\_CONTEXT\_DATA

This structure holds the integrity value and the encrypted data for a context.

NOTE This is an informative table that is included in the specification only to allow calculation of the maximum size for TPM2B\_CONTEXT\_DATA.

Table 219 — Definition of TPMS\_CONTEXT\_DATA Structure <IN/OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| integrity | TPM2B\_DIGEST | the integrity value |
| encrypted | TPM2B\_CONTEXT\_SENSITIVE | the sensitive area |

## TPM2B\_CONTEXT\_DATA

This structure is used in a TPMS\_CONTEXT.

Table 220 — Definition of TPM2B\_CONTEXT\_DATA Structure <IN/OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size | UINT16 |  |
| buffer[size] {:sizeof(TPMS\_CONTEXT\_DATA)} | BYTE |  |

## TPMS\_CONTEXT

This structure is used in TPM2\_ContextLoad() and TPM2\_ContextSave(). If the values of the TPMS\_CONTEXT structure in TPM2\_ContextLoad() are not the same as the values when the context was saved (TPM2\_ContextSave()), then the TPM shall not load the context.

Saved object contexts shall not be loaded as long as the associated hierarchy is disabled.

Saved object contexts are invalidated when the Primary Seed of their hierarchy changes. Objects in the Endorsement hierarchy are invalidated when either the EPS or SPS is changed.

When an object has the *stClear* attribute, it shall not be possible to reload the context or any descendant object after a TPM Reset or TPM Restart.

NOTE 1 The reference implementation prevents reloads after TPM Restart by including the current value of a *clearCount* in the saved object context. When an object is loaded, this value is compared with the current value of the *clearCount* if the object has the *stClear* attribute. If the values are not the same, then the object cannot be loaded.

A sequence value is contained within *contextBlob*, the integrity-protected part of the saved context. The sequence value is repeated in the *sequence* parameter of the TPMS\_CONTEXT structure. The *sequence* parameter, along with other values, is used in the generation the protection values of the context.

NOTE 2 The reference implementation prepends the *sequence* value to the *contextBlob* before, for example, the SESSION structure for sessions or the OBJECT structure for transient objects.

If the integrity value of the context is valid, but the *sequence* value of the decrypted context does not match the value in the *sequence* parameter, then TPM shall enter the failure mode because this is indicative of a specific type of attack on the context values.

NOTE 3 If the integrity value is correct, but the decryption fails and produces the wrong value for sequence, this implies that either the TPM is faulty or an external entity is able to forge an integrity value for the context but they have insufficient information to know the encryption key of the context. Since the TPM generated the valid context, then there is no reason for the sequence value in the context to be decrypted incorrectly other than the TPM is faulty or the TPM is under attack. In either case, it is appropriate for the TPM to enter failure more.

Table 221 — Definition of TPMS\_CONTEXT Structure

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| sequence | UINT64 | the sequence number of the context  NOTE Transient object contexts and session contexts used different counters. |
| savedHandle | TPMI\_DH\_SAVED | a handle indicating if the context is a session, object, or sequence object (see Table 222 — Context Handle Values |
| hierarchy | TPMI\_RH\_HIERARCHY+ | the hierarchy of the context |
| contextBlob | TPM2B\_CONTEXT\_DATA | the context data and integrity HMAC |

## Parameters of TPMS\_CONTEXT

### *sequence*

The *sequence* parameter is used to differentiate the contexts and to allow the TPM to create a different encryption key for each context. Objects and sessions use different sequence counters. The sequence counter for objects (transient and sequence) is incremented when an object context is saved, and the sequence counter for sessions increments when a session is created or when it is loaded (TPM2\_ContextLoad()). The session sequence number is the *contextID* counter.

For a session, the sequence number also allows the TRM to find the “older” contexts so that they may be refreshed if the *contextID* are too widely separated.

If an input value for *sequence* is larger than the value used in any saved context, the TPM shall return an error (TPM\_RC\_VALUE) and do no additional processing of the context.

If the context is a session context and the input value for sequence is less than the current value of *contextID* minus the maximum range for sessions, the TPM shall return an error (TPM\_RC\_VALUE) and do no additional processing of the context.

### *savedHandle*

For a session, this is the handle that was assigned to the session when it was created. For a transient object, the handle will have one of the values shown in Table 222.

If the handle type for *savedHandle* is TPM\_HT\_TRANSIENT, then the low order bits are used to differentiate static objects from sequence objects.

If an input value for handle is outside of the range of values used by the TPM, the TPM shall return an error (TPM\_RC\_VALUE) and do no additional processing of the context.

Table 222 — Context Handle Values

|  |  |
| --- | --- |
| Value | Description |
| 0x02xxxxxx | an HMAC session context |
| 0x03xxxxxx | a policy session context |
| 0x80000000 | an ordinary transient object |
| 0x80000001 | a sequence object |
| 0x80000002 | a transient object with the *stClear* attribute SET |

### *hierarchy*

This is the hierarchy (TPMI\_RH\_HIERARCHY) for the saved context and determines the proof value used in the construction of the encryption and integrity values for the context. For session and sequence contexts, the hierarchy is TPM\_RC\_NULL. The hierarchy for a transient object may be TPM\_RH\_NULL but it is not required.

## Context Protection

### Context Integrity

The integrity of the context blob is protected by an HMAC. The integrity value is constructed such that changes to the component values will invalidate the context and prevent it from being loaded.

Previously saved contexts for objects in the Platform hierarchy shall not be loadable after the PPS is changed.

Previously saved contexts for objects in the Storage hierarchy shall not be loadable after the SPS is changed.

Previously saved contexts for objects in the Endorsement hierarchy shall not be loadable after either the EPS or SPS is changed.

Previously saved sessions shall not be loadable after the SPS changes.

Previously saved contexts for objects that have their *stClear* attribute SET shall not be loadable after a TPM Restart. If a Storage Key has its *stClear* attribute SET, the descendants of this key shall not be loadable after TPM Restart.

Previously saved contexts for a session and objects shall not be loadable after a TPM Reset.

A saved context shall not be loaded if its HMAC is not valid. The equation for computing the HMAC for a context is found in “Context Integrity Protection” in TPM 2.0 Part 1.

### Context Confidentiality

The context data of sessions and objects shall be protected by symmetric encryption using CFB. The method for computing the IV and encryption key is found in “Context Confidentiality Protection” in TPM 2.0 Part 1.

# Creation Data

## TPMS\_CREATION\_DATA

This structure provides information relating to the creation environment for the object. The creation data includes the parent Name, parent Qualified Name, and the digest of selected PCR. These values represent the environment in which the object was created. Creation data allows a relying party to determine if an object was created when some appropriate protections were present.

When the object is created, the structure shown in Table 223 is generated and a ticket is computed over this data.

If the parent is a permanent handle (TPM\_RH\_OWNER, TPM\_RH\_PLATFORM, TPM\_RH\_ENDORSEMENT, or TPM\_RH\_NULL), then *parentName* and *parentQualifiedName* will be set to the parent handle value and *parentNameAlg* will be TPM\_ALG\_NULL.

Table 223 — Definition of TPMS\_CREATION\_DATA Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| pcrSelect | TPML\_PCR\_SELECTION | list indicating the PCR included in *pcrDigest* |
| pcrDigest | TPM2B\_DIGEST | digest of the selected PCR using *nameAlg* of the object for which this structure is being created  *pcrDigest.size* shall be zero if the *pcrSelect* list is empty. |
| locality | TPMA\_LOCALITY | the locality at which the object was created |
| parentNameAlg | TPM\_ALG\_ID | *nameAlg* of the parent |
| parentName | TPM2B\_NAME | Name of the parent at time of creation  The size will match digest size associated with *parentNameAlg* unless it is TPM\_ALG\_NULL, in which case the size will be 4 and *parentName* will be the hierarchy handle. |
| parentQualifiedName | TPM2B\_NAME | Qualified Name of the parent at the time of creation  Size is the same as *parentName.* |
| outsideInfo | TPM2B\_DATA | association with additional information added by the key creator  This will be the contents of the *outsideInfo* parameter in TPM2\_Create() or TPM2\_CreatePrimary(). |

## TPM2B\_CREATION\_DATA

This structure is created by TPM2\_Create() and TPM2\_CreatePrimary(). It is never entered into the TPM and never has a size of zero.

Table 224 — Definition of TPM2B\_CREATION\_DATA Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| size= | UINT16 | size of the creation data |
| creationData | TPMS\_CREATION\_DATA |  |

# Attached Component Structures

## TPM\_AT

These constants are used in TPM2\_AC\_GetCapability() to indicate the first tagged value returned from an attached component.

TPM\_AT values of 0x80000000 through 0xFFFFFFFF are reserved for vendor-specific values.

Table 225 — Definition of (UINT32) TPM\_AT Constants

| Name | Value | Comments |
| --- | --- | --- |
| TPM\_AT\_ANY | 0x00000000 | in a command, a non-specific request for AC information; in a response, indicates that *outputData* is not meaningful |
| TPM\_AT\_ERROR | 0x00000001 | indicates a TCG defined, device-specific error |
| TPM\_AT\_PV1 | 0x00000002 | indicates the most significant 32 bits of a pairing value for the AC |
| TPM\_AT\_VEND | 0x80000000 | value added to a TPM\_AT to indicate a vendor-specific tag value |

## TPM\_AE

These constants are the TCG-defined error values returned by an AC.

Table 226 — Definition of (UINT32) TPM\_AE Constants <OUT>

| Name | Value | Comments |
| --- | --- | --- |
| TPM\_AE\_NONE | 0x00000000 | in a command, a non-specific request for AC information; in a response, indicates that *outputData* is not meaningful |

## TPMS\_AC\_OUTPUT

TPMS\_AC\_OUTPUT is used to return information about an AC. The *tag* structure parameter indicates the type of the *data* value*.*

Table 227 — Definition of TPMS\_AC\_OUTPUT Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| tag | TPM\_AT | tag indicating the contents of *data* |
| data | UINT32 | the data returned from the AC |

## TPML\_AC\_CAPABILITIES

This list is only used in TPM2\_AC\_GetCapability().

The values in the list are returned in TPM\_AT order (see Table 225) with vendor-specific values returned after TCG defined values.

NOTE MAX\_AC\_CAPABILITIES = MAX\_CAP\_DATA / sizeof(TPMS\_AC\_OUTPUT)

Table 228 — Definition of TPML\_AC\_CAPABILITIES Structure <OUT>

|  |  |  |
| --- | --- | --- |
| Parameter | Type | Description |
| count | UINT32 | number of values in the *acCapabilities* list; may be 0 |
| acCapabilities[count] {:MAX\_AC\_CAPABILITIES} | TPMS\_AC\_OUTPUT | a list of AC values |