

Trusted Platform Module 2.0 Library Part 2: Structures

Version 185
March 12, 2026

Contact: admin@trustedcomputinggroup.org

TCG Published

DISCLAIMERS, NOTICES, AND LICENSE TERMS

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) 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.

DOCUMENT STYLE

Key Words

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in this document’s normative statements are to be interpreted as described in [RFC 2119: Key words for use in RFCs to Indicate Requirement Levels](#).

Statement Type

Please note an important distinction between different sections of text throughout this document. There are two distinctive kinds of text: *informative comments* and *normative statements*. Because most of the text in this specification will be of the kind *normative statements*, the authors have informally defined it as the default and, as such, have specifically called out text of the kind *informative comment*. They have done this by flagging the beginning and end of each informative comment and highlighting its text in gray. This means that unless text is specifically marked as of the kind *informative comment*, it can be considered a *normative statement*.

Example:

Reach out to admin@trustedcomputinggroup with any questions about this document.

Contents

| | | |
|----------|---|-----------|
| 1 | Scope | 17 |
| 2 | Terms and definitions | 18 |
| 3 | Symbols and Abbreviated Terms | 19 |
| 4 | Notation | 20 |
| 4.1 | Introduction | 20 |
| 4.2 | Named Constants | 20 |
| 4.3 | Data Type Aliases (typedefs) | 20 |
| 4.4 | Enumerations | 21 |
| 4.5 | Interface Type | 22 |
| 4.6 | Arrays | 23 |
| 4.7 | Structure Definitions | 24 |
| 4.8 | Conditional Types | 25 |
| 4.9 | Unions | 27 |
| 4.9.1 | Introduction | 27 |
| 4.9.2 | Union Definition | 27 |
| 4.9.3 | Union Instance | 29 |
| 4.9.4 | Union Selector Definition | 30 |
| 4.10 | Bit Field Definitions | 31 |
| 4.11 | Parameter Limits | 32 |
| 4.12 | Size Checking | 33 |
| 4.13 | Name Prefix Convention | 34 |
| 4.14 | Data Alignment | 36 |
| 4.15 | Parameter Unmarshaling Errors | 36 |
| 5 | Base Types | 38 |
| 5.1 | Primitive Types | 38 |
| 5.2 | Specification Logic Value Constants | 38 |
| 5.3 | Miscellaneous Types | 39 |
| 5.3.1 | Key Size Types | 39 |
| 6 | Constants | 40 |
| 6.1 | TPM_SPEC (Specification Version Values) | 40 |
| 6.2 | TPM_CONSTANTS32 | 40 |
| 6.3 | TPM_ALG_ID | 41 |
| 6.4 | TPM_ECC_CURVE | 46 |
| 6.5 | TPM_CC | 46 |
| 6.5.1 | Format | 46 |
| 6.5.2 | TPM_CC Listing | 47 |
| 6.6 | TPM_RC | 53 |
| 6.6.1 | Description | 53 |
| 6.6.2 | Response Code Formats | 54 |
| 6.6.3 | TPM_RC Values | 57 |
| 6.7 | TPM_CLOCK_ADJUST | 64 |
| 6.8 | TPM_EO | 64 |
| 6.9 | TPM_ST | 65 |
| 6.10 | TPM_SU | 68 |
| 6.11 | TPM_SE | 69 |

| | | |
|----------|------------------------------|------------|
| 6.12 | TPM_CAP | 69 |
| 6.12.1 | Settable Capabilities | 70 |
| 6.13 | TPM_PT | 72 |
| 6.14 | TPM_PT_PCR | 81 |
| 6.15 | TPM_PS | 84 |
| 6.16 | TPM_PUB_KEY | 85 |
| 7 | Handles | 86 |
| 7.1 | Introduction | 86 |
| 7.2 | TPM_HT | 86 |
| 7.3 | Persistent Handle Sub-ranges | 88 |
| 7.4 | TPM_RH | 88 |
| 7.5 | TPM_HC | 90 |
| 8 | Attribute Structures | 94 |
| 8.1 | Description | 94 |
| 8.2 | TPMA_ALGORITHM | 94 |
| 8.3 | TPMA_OBJECT | 95 |
| 8.3.1 | Introduction | 95 |
| 8.3.2 | Structure Definition | 95 |
| 8.3.3 | Attribute Descriptions | 97 |
| 8.4 | TPMA_SESSION | 104 |
| 8.5 | TPMA_LOCALITY | 106 |
| 8.6 | TPMA_PERMANENT | 107 |
| 8.7 | TPMA_STARTUP_CLEAR | 108 |
| 8.8 | TPMA_MEMORY | 111 |
| 8.9 | TPMA_CC | 112 |
| 8.9.1 | Introduction | 112 |
| 8.9.2 | Structure Definition | 112 |
| 8.9.3 | Field Descriptions | 112 |
| 8.10 | TPMA_MODES | 114 |
| 8.11 | TPMA_X509_KEY_USAGE | 115 |
| 8.12 | TPMA_ACT | 116 |
| 8.13 | TPMA_ML_PARAMETER_SET | 117 |
| 9 | Interface Types | 118 |
| 9.1 | Introduction | 118 |
| 9.2 | TPMI_YES_NO | 118 |
| 9.3 | TPMI_DH_OBJECT | 118 |
| 9.4 | TPMI_DH_PARENT | 118 |
| 9.5 | TPMI_DH_PERSISTENT | 119 |
| 9.6 | TPMI_DH_ENTITY | 119 |
| 9.7 | TPMI_DH_PCR | 120 |
| 9.8 | TPMI_SH_AUTH_SESSION | 120 |
| 9.9 | TPMI_SH_HMAC | 121 |
| 9.10 | TPMI_SH_POLICY | 121 |
| 9.11 | TPMI_DH_CONTEXT | 121 |
| 9.12 | TPMI_DH_SAVED | 121 |
| 9.13 | TPMI_RH_HIERARCHY | 122 |
| 9.14 | TPMI_RH_ENABLES | 122 |
| 9.15 | TPMI_RH_HIERARCHY_AUTH | 123 |
| 9.16 | TPMI_RH_HIERARCHY_POLICY | 123 |

| | | |
|-----------|------------------------------|------------|
| 9.17 | TPMI_RH_BASE_HIERARCHY | 123 |
| 9.18 | TPMI_RH_PLATFORM | 124 |
| 9.19 | TPMI_RH_OWNER | 124 |
| 9.20 | TPMI_RH_ENDORSEMENT | 124 |
| 9.21 | TPMI_RH_PROVISION | 124 |
| 9.22 | TPMI_RH_CLEAR | 125 |
| 9.23 | TPMI_RH_NV_AUTH | 125 |
| 9.24 | TPMI_RH_LOCKOUT | 125 |
| 9.25 | TPMI_RH_NV_INDEX | 126 |
| 9.26 | TPMI_RH_NV_DEFINED_INDEX | 126 |
| 9.27 | TPMI_RH_NV_LEGACY_INDEX | 126 |
| 9.28 | TPMI_RH_NV_EXP_INDEX | 126 |
| 9.29 | TPMI_RH_AC | 127 |
| 9.30 | TPMI_RH_ACT | 127 |
| 9.31 | TPMI_ALG_HASH | 127 |
| 9.32 | TPMI_ALG_ASYM | 128 |
| 9.33 | TPMI_ALG_SYM | 128 |
| 9.34 | TPMI_ALG_SYM_OBJECT | 129 |
| 9.35 | TPMI_ALG_SYM_MODE | 129 |
| 9.36 | TPMI_ALG_KDF | 130 |
| 9.37 | TPMI_ALG_SIG_SCHEME | 130 |
| 9.38 | TPMI_ECC_KEY_EXCHANGE | 131 |
| 9.39 | TPMI_ST_COMMAND_TAG | 131 |
| 9.40 | TPMI_ALG_MAC_SCHEME | 132 |
| 9.41 | TPMI_ALG_CIPHER_MODE | 132 |
| 10 | Structure Definitions | 134 |
| 10.1 | TPMS_EMPTY | 134 |
| 10.2 | Hash/Digest Structures | 134 |
| 10.2.1 | TPMU_HA | 134 |
| 10.2.2 | TPMT_HA | 134 |
| 10.3 | Sized Buffers | 135 |
| 10.3.1 | Introduction | 135 |
| 10.3.2 | TPM2B_DIGEST | 136 |
| 10.3.3 | TPM2B_DATA | 136 |
| 10.3.4 | TPM2B_NONCE | 136 |
| 10.3.5 | TPM2B_AUTH | 137 |
| 10.3.6 | TPM2B_OPERAND | 137 |
| 10.3.7 | TPM2B_EVENT | 137 |
| 10.3.8 | TPM2B_MAX_BUFFER | 137 |
| 10.3.9 | TPM2B_MAX_NV_BUFFER | 138 |
| 10.3.10 | TPM2B_TIMEOUT | 138 |
| 10.3.11 | TPM2B_IV | 138 |
| 10.3.12 | TPM2B_SHARED_SECRET | 139 |
| 10.3.13 | TPMU_KEM_CIPHERTEXT | 139 |
| 10.3.14 | TPM2B_KEM_CIPHERTEXT | 140 |
| 10.3.15 | TPM2B_VENDOR_PROPERTY | 140 |
| 10.4 | Names | 140 |
| 10.4.1 | Introduction | 140 |
| 10.4.2 | TPMU_NAME | 141 |
| 10.4.3 | TPM2B_NAME | 141 |

| | | |
|---------|--------------------------------------|-----|
| 10.5 | PCR Structures | 141 |
| 10.5.1 | TPMS_PCR_SELECT | 141 |
| 10.5.2 | TPMS_PCR_SELECTION | 142 |
| 10.6 | Tickets | 143 |
| 10.6.1 | Introduction | 143 |
| 10.6.2 | A NULL Ticket | 144 |
| 10.6.3 | TPMT_TK_CREATION | 144 |
| 10.6.4 | TPMU_TK_VERIFIED_META | 145 |
| 10.6.5 | TPMT_TK_VERIFIED | 145 |
| 10.6.6 | TPMT_TK_AUTH | 146 |
| 10.6.7 | TPMT_TK_HASHCHECK | 148 |
| 10.7 | Property Structures | 148 |
| 10.7.1 | TPMS_ALG_PROPERTY | 148 |
| 10.7.2 | TPMS_TAGGED_PROPERTY | 148 |
| 10.7.3 | TPMS_TAGGED_PCR_SELECT | 149 |
| 10.7.4 | TPMS_TAGGED_POLICY | 149 |
| 10.7.5 | TPMS_ACT_DATA | 149 |
| 10.7.6 | TPMS_SPDM_SESSION_INFO | 149 |
| 10.8 | Lists | 150 |
| 10.8.1 | TPML_CC | 150 |
| 10.8.2 | TPML_CCA | 150 |
| 10.8.3 | TPML_ALG | 150 |
| 10.8.4 | TPML_HANDLE | 151 |
| 10.8.5 | TPML_DIGEST | 151 |
| 10.8.6 | TPML_DIGEST_VALUES | 152 |
| 10.8.7 | TPML_PCR_SELECTION | 152 |
| 10.8.8 | TPML_ALG_PROPERTY | 153 |
| 10.8.9 | TPML_TAGGED_TPM_PROPERTY | 153 |
| 10.8.10 | TPML_TAGGED_PCR_PROPERTY | 153 |
| 10.8.11 | TPML_ECC_CURVE | 154 |
| 10.8.12 | TPML_TAGGED_POLICY | 154 |
| 10.8.13 | TPML_ACT_DATA | 155 |
| 10.8.14 | TPML_PUB_KEY | 155 |
| 10.8.15 | TPML_SPDM_SESSION_INFO | 155 |
| 10.8.16 | TPML_VENDOR_PROPERTY | 156 |
| 10.9 | Capabilities Structures | 156 |
| 10.9.1 | TPMU_CAPABILITIES | 157 |
| 10.9.2 | TPMS_CAPABILITY_DATA | 157 |
| 10.9.3 | TPMS_SET_CAPABILITY_DATA | 157 |
| 10.9.4 | TPM2B_SET_CAPABILITY_DATA | 158 |
| 10.10 | Clock/Counter Structures | 158 |
| 10.10.1 | TPMS_CLOCK_INFO | 158 |
| 10.10.2 | <i>Clock</i> | 158 |
| 10.10.3 | <i>ResetCount</i> | 159 |
| 10.10.4 | <i>RestartCount</i> | 159 |
| 10.10.5 | <i>Safe</i> | 159 |
| 10.10.6 | TPMS_TIME_INFO | 159 |
| 10.11 | TPM Attestation Structures | 159 |
| 10.11.1 | Introduction | 159 |
| 10.11.2 | TPMS_TIME_ATTEST_INFO | 160 |
| 10.11.3 | TPMS_CERTIFY_INFO | 160 |

| | | |
|-----------|--|------------|
| 10.11.4 | TPMS_QUOTE_INFO | 160 |
| 10.11.5 | TPMS_COMMAND_AUDIT_INFO | 160 |
| 10.11.6 | TPMS_SESSION_AUDIT_INFO | 161 |
| 10.11.7 | TPMS_CREATION_INFO | 161 |
| 10.11.8 | TPMS_NV_CERTIFY_INFO | 161 |
| 10.11.9 | TPMS_NV_DIGEST_CERTIFY_INFO | 161 |
| 10.11.10 | TPMI_ST_ATTEST | 162 |
| 10.11.11 | TPMU_ATTEST | 162 |
| 10.11.12 | TPMS_ATTEST | 162 |
| 10.11.13 | TPM2B_ATTEST | 163 |
| 10.12 | Authorization Structures | 163 |
| 10.12.1 | Introduction | 163 |
| 10.12.2 | TPMS_AUTH_COMMAND | 163 |
| 10.12.3 | TPMS_AUTH_RESPONSE | 164 |
| 11 | Algorithm Parameters and Structures | 165 |
| 11.1 | Symmetric | 165 |
| 11.1.1 | Introduction | 165 |
| 11.1.2 | Symmetric Cipher Key Sizes | 165 |
| 11.1.3 | TPMU_SYM_KEY_BITS | 165 |
| 11.1.4 | TPMU_SYM_MODE | 166 |
| 11.1.5 | TPMU_SYM_DETAILS | 166 |
| 11.1.6 | TPMT_SYM_DEF | 167 |
| 11.1.7 | TPMT_SYM_DEF_OBJECT | 167 |
| 11.1.8 | TPM2B_SYM_KEY | 168 |
| 11.1.9 | TPMS_SYMCIPHER_PARMS | 168 |
| 11.1.10 | TPM2B_LABEL | 168 |
| 11.1.11 | TPMS_DERIVE | 169 |
| 11.1.12 | TPM2B_DERIVE | 169 |
| 11.1.13 | TPMU_SENSITIVE_CREATE | 169 |
| 11.1.14 | TPM2B_SENSITIVE_DATA | 170 |
| 11.1.15 | TPMS_SENSITIVE_CREATE | 170 |
| 11.1.16 | TPM2B_SENSITIVE_CREATE | 170 |
| 11.1.17 | TPMS_SCHEME_HASH | 171 |
| 11.1.18 | TPMS_SCHEME_ECDSA | 171 |
| 11.1.19 | TPMI_ALG_KEYEDHASH_SCHEME | 171 |
| 11.1.20 | TPMS_SCHEME_HMAC | 172 |
| 11.1.21 | TPMS_SCHEME_XOR | 172 |
| 11.1.22 | TPMU_SCHEME_KEYEDHASH | 172 |
| 11.1.23 | TPMT_KEYEDHASH_SCHEME | 172 |
| 11.2 | Asymmetric | 173 |
| 11.2.1 | Signing Schemes | 173 |
| 11.2.2 | Encryption Schemes | 174 |
| 11.2.3 | Key Derivation Schemes | 175 |
| 11.2.4 | RSA | 176 |
| 11.2.5 | ECC | 179 |
| 11.2.6 | ML-KEM | 182 |
| 11.2.7 | ML-DSA | 183 |
| 11.3 | Signatures | 185 |
| 11.3.1 | TPMS_SIGNATURE_RSA | 185 |
| 11.3.2 | TPMS_SIGNATURE_ECC | 185 |

| | | |
|-----------|--------------------------------|------------|
| 11.3.3 | TPM2B_SIGNATURE_EDDSA | 186 |
| 11.3.4 | TPM2B_SIGNATURE_MLDSA | 186 |
| 11.3.5 | TPMU_SIGNATURE | 186 |
| 11.3.6 | TPMT_SIGNATURE | 187 |
| 11.3.7 | TPMU_SIGNATURE_CTX | 187 |
| 11.3.8 | TPM2B_SIGNATURE_CTX | 188 |
| 11.3.9 | TPM2B_SIGNATURE_HINT | 188 |
| 11.4 | Key/Secret Exchange | 189 |
| 11.4.1 | Introduction | 189 |
| 11.4.2 | TPMU_ENCRYPTED_SECRET | 189 |
| 11.4.3 | TPM2B_ENCRYPTED_SECRET | 190 |
| 12 | Key/Object Complex | 191 |
| 12.1 | Introduction | 191 |
| 12.2 | Public Area Structures | 191 |
| 12.2.1 | Description | 191 |
| 12.2.2 | TPMI_ALG_PUBLIC | 191 |
| 12.2.3 | Type-Specific Parameters | 191 |
| 12.2.4 | TPMT_PUBLIC | 198 |
| 12.2.5 | TPM2B_PUBLIC | 199 |
| 12.2.6 | TPM2B_TEMPLATE | 200 |
| 12.3 | Private Area Structures | 200 |
| 12.3.1 | Introduction | 200 |
| 12.3.2 | Sensitive Data Structures | 200 |
| 12.3.3 | TPM2B_SENSITIVE | 202 |
| 12.3.4 | Encryption | 202 |
| 12.3.5 | Integrity | 202 |
| 12.3.6 | _PRIVATE | 202 |
| 12.3.7 | TPM2B_PRIVATE | 203 |
| 12.4 | Identity Object | 203 |
| 12.4.1 | Description | 203 |
| 12.4.2 | TPMS_ID_OBJECT | 203 |
| 12.4.3 | TPM2B_ID_OBJECT | 204 |
| 13 | NV Storage Structures | 205 |
| 13.1 | TPM_NV_INDEX | 205 |
| 13.2 | TPM_NT | 205 |
| 13.3 | TPMS_NV_PIN_COUNTER_PARAMETERS | 206 |
| 13.4 | TPMA_NV | 206 |
| 13.5 | TPMA_NV_EXP | 210 |
| 13.6 | TPMS_NV_PUBLIC | 211 |
| 13.7 | TPM2B_NV_PUBLIC | 212 |
| 13.8 | TPMS_NV_PUBLIC_EXP_ATTR | 212 |
| 13.9 | TPMU_NV_PUBLIC_2 | 213 |
| 13.10 | TPMT_NV_PUBLIC_2 | 214 |
| 13.11 | TPM2B_NV_PUBLIC_2 | 214 |
| 14 | Context Data | 215 |
| 14.1 | Introduction | 215 |
| 14.2 | TPM2B_CONTEXT_SENSITIVE | 215 |
| 14.3 | TPMS_CONTEXT_DATA | 215 |
| 14.4 | TPM2B_CONTEXT_DATA | 215 |

| | | |
|-----------|--------------------------------------|------------|
| 14.5 | TPMS_CONTEXT | 216 |
| 14.6 | Parameters of TPMS_CONTEXT | 217 |
| 14.6.1 | <i>sequence</i> | 217 |
| 14.6.2 | <i>savedHandle</i> | 217 |
| 14.6.3 | <i>hierarchy</i> | 218 |
| 14.7 | Context Protection | 218 |
| 14.7.1 | Context Integrity | 218 |
| 14.7.2 | Context Confidentiality | 218 |
| 15 | Creation Data | 219 |
| 15.1 | TPMS_CREATION_DATA | 219 |
| 15.2 | TPM2B_CREATION_DATA | 219 |
| 16 | Attached Component Structures | 221 |
| 16.1 | TPM_AT | 221 |
| 16.2 | TPM_AE | 221 |
| 16.3 | TPMS_AC_OUTPUT | 221 |
| 16.4 | TPML_AC_CAPABILITIES | 221 |
| | References | 223 |

List of Tables

| | | |
|----|---|-----|
| 1 | Name Prefix Convention | 35 |
| 2 | Unmarshaling Errors | 37 |
| 3 | Definition of Base Types | 38 |
| 4 | Defines for Logic Values | 38 |
| 5 | Definition of Types for TPM_KEY_BITS | 39 |
| 6 | Definition of (UINT32) TPM_SPEC Constants | 40 |
| 7 | Definition of (UINT32) TPM_CONSTANTS32 Constants | 41 |
| 8 | Definition of (UINT16) TPM_ALG_ID Constants | 41 |
| 9 | Definition of (UINT16) (ECC) TPM_ECC_CURVE Constants | 46 |
| 10 | TPM Command (TPM_CC) Format | 47 |
| 11 | TPM Command (TPM_CC) Fields Description | 47 |
| 12 | Definition of (UINT32) TPM_CC Constants (Numeric Order) | 47 |
| 13 | Format-Zero Response Codes (TPM_RC) | 54 |
| 14 | Format-Zero Response Codes (TPM_RC) Fields Description | 54 |
| 15 | Format-One Response Codes (TPM_RC) | 55 |
| 16 | Format-One Response Codes (TPM_RC) Fields Description | 55 |
| 17 | Response Code Groupings | 56 |
| 18 | Definition of (UINT32) TPM_RC Constants (Actions) | 57 |
| 19 | Definition of (INT8) TPM_CLOCK_ADJUST Constants | 64 |
| 20 | Definition of (UINT16) TPM_EO Constants | 64 |
| 21 | Definition of (UINT16) TPM_ST Constants | 65 |
| 22 | Definition of (UINT16) TPM_SU Constants | 69 |
| 23 | Definition of (UINT8) TPM_SE Constants | 69 |
| 24 | Definition of (UINT32) TPM_CAP Constants | 69 |
| 25 | TPM Capability (TPM_CAP) Format | 71 |
| 26 | TPM Capability (TPM_CAP) Fields Description | 71 |
| 27 | TPM Capability Authorization | 72 |
| 28 | Definition of (UINT32) TPM_PT Constants | 72 |
| 29 | Definition of (UINT32) TPM_PT_PCR Constants | 82 |
| 30 | Definition of (UINT32) TPM_PS Constants | 84 |
| 31 | Definition of (UINT32) TPM_PUB_KEY Constants | 85 |
| 32 | Definition of Types for TPM_HANDLE | 86 |
| 33 | Definition of (UINT8) TPM_HT Constants | 86 |
| 34 | Definition of (TPM_HANDLE) TPM_RH Constants | 88 |
| 35 | Definition of (TPM_HANDLE) TPM_HC Constants | 91 |
| 36 | Definition of (UINT32) TPMA_ALGORITHM Bits | 94 |
| 37 | Definition of (UINT32) TPMA_OBJECT Bits | 95 |
| 38 | Definition of (UINT8) TPMA_SESSION Bits | 104 |
| 39 | Definition of (UINT8) TPMA_LOCALITY Bits | 107 |
| 40 | Definition of (UINT32) TPMA_PERMANENT Bits | 107 |
| 41 | Definition of (UINT32) TPMA_STARTUP_CLEAR Bits | 108 |
| 42 | Definition of (UINT32) TPMA_MEMORY Bits | 111 |
| 43 | Definition of (UINT32) TPMA_CC Bits | 112 |
| 44 | Definition of (UINT32) TPMA_MODES Bits | 114 |
| 45 | Definition of (UINT32) TPMA_X509_KEY_USAGE Bits | 116 |
| 46 | Definition of (UINT32) TPMA_ACT Bits | 116 |
| 47 | Definition of (UINT32) TPMA_ML_PARAMETER_SET Bits | 117 |
| 48 | Definition of (BYTE) TPMI_YES_NO Type | 118 |
| 49 | Definition of (TPM_HANDLE) TPMI_DH_OBJECT Type | 118 |
| 50 | Definition of (TPM_HANDLE) TPMI_DH_PARENT Type | 119 |

| | | |
|-----|---|-----|
| 51 | Definition of (TPM_HANDLE) TPMI_DH_PERSISTENT Type | 119 |
| 52 | Definition of (TPM_HANDLE) TPMI_DH_ENTITY Type | 119 |
| 53 | Definition of (TPM_HANDLE) TPMI_DH_PCR Type | 120 |
| 54 | Definition of (TPM_HANDLE) TPMI_SH_AUTH_SESSION Type | 120 |
| 55 | Definition of (TPM_HANDLE) TPMI_SH_HMAC Type | 121 |
| 56 | Definition of (TPM_HANDLE) TPMI_SH_POLICY Type | 121 |
| 57 | Definition of (TPM_HANDLE) TPMI_DH_CONTEXT Type | 121 |
| 58 | Definition of (TPM_HANDLE) TPMI_DH_SAVED Type | 121 |
| 59 | Definition of (TPM_HANDLE) TPMI_RH_HIERARCHY Type | 122 |
| 60 | Definition of (TPM_HANDLE) TPMI_RH_ENABLES Type | 122 |
| 61 | Definition of (TPM_HANDLE) TPMI_RH_HIERARCHY_AUTH Type | 123 |
| 62 | Definition of (TPM_HANDLE) TPMI_RH_HIERARCHY_POLICY Type | 123 |
| 63 | Definition of (TPM_HANDLE) TPMI_RH_BASE_HIERARCHY Type | 124 |
| 64 | Definition of (TPM_HANDLE) TPMI_RH_PLATFORM Type | 124 |
| 65 | Definition of (TPM_HANDLE) TPMI_RH_OWNER Type | 124 |
| 66 | Definition of (TPM_HANDLE) TPMI_RH_ENDORSEMENT Type | 124 |
| 67 | Definition of (TPM_HANDLE) TPMI_RH_PROVISION Type | 125 |
| 68 | Definition of (TPM_HANDLE) TPMI_RH_CLEAR Type | 125 |
| 69 | Definition of (TPM_HANDLE) TPMI_RH_NV_AUTH Type | 125 |
| 70 | Definition of (TPM_HANDLE) TPMI_RH_LOCKOUT Type | 126 |
| 71 | Definition of (TPM_HANDLE) TPMI_RH_NV_INDEX Type | 126 |
| 72 | Definition of (TPM_HANDLE) TPMI_RH_NV_DEFINED_INDEX Type | 126 |
| 73 | Definition of (TPM_HANDLE) TPMI_RH_NV_LEGACY_INDEX Type | 126 |
| 74 | Definition of (TPM_HANDLE) TPMI_RH_NV_EXP_INDEX Type | 127 |
| 75 | Definition of (TPM_HANDLE) TPMI_RH_AC Type | 127 |
| 76 | Definition of (TPM_HANDLE) TPMI_RH_ACT Type | 127 |
| 77 | Definition of (TPM_ALG_ID) TPMI_ALG_HASH Type | 127 |
| 78 | Definition of (TPM_ALG_ID) TPMI_ALG_ASYM Type | 128 |
| 79 | Definition of (TPM_ALG_ID) TPMI_ALG_SYM Type | 129 |
| 80 | Definition of (TPM_ALG_ID) TPMI_ALG_SYM_OBJECT Type | 129 |
| 81 | Definition of (TPM_ALG_ID) TPMI_ALG_SYM_MODE Type | 130 |
| 82 | Definition of (TPM_ALG_ID) TPMI_ALG_KDF Type | 130 |
| 83 | Definition of (TPM_ALG_ID) TPMI_ALG_SIG_SCHEME Type | 130 |
| 84 | Definition of (TPM_ALG_ID) (ECC) TPMI_ECC_KEY_EXCHANGE Type | 131 |
| 85 | Definition of (TPM_ST) TPMI_ST_COMMAND_TAG Type | 132 |
| 86 | Definition of (TPM_ALG_ID) TPMI_ALG_MAC_SCHEME Type | 132 |
| 87 | Definition of (TPM_ALG_ID) TPMI_ALG_CIPHER_MODE Type | 133 |
| 88 | Definition of TPMU_HA Union | 134 |
| 89 | Definition of TPMT_HA Structure | 134 |
| 90 | Definition of TPM2B_DIGEST Structure | 136 |
| 91 | Definition of TPM2B_DATA Structure | 136 |
| 92 | Definition of Types for TPM2B_NONCE | 136 |
| 93 | Definition of Types for TPM2B_AUTH | 137 |
| 94 | Definition of Types for TPM2B_OPERAND | 137 |
| 95 | Definition of TPM2B_EVENT Structure | 137 |
| 96 | Definition of TPM2B_MAX_BUFFER Structure | 138 |
| 97 | Definition of TPM2B_MAX_NV_BUFFER Structure | 138 |
| 98 | Definition of TPM2B_TIMEOUT Structure | 138 |
| 99 | Definition of TPM2B_IV Structure | 139 |
| 100 | Definition of TPM2B_SHARED_SECRET Structure | 139 |
| 101 | Definition of TPMU_KEM_CIPHERTEXT Union | 140 |

| | | |
|-----|---|-----|
| 102 | Definition of TPM2B_KEM_CIPHERTEXT Structure | 140 |
| 103 | Definition of TPM2B_VENDOR_PROPERTY Structure | 140 |
| 104 | Definition of TPMU_NAME Union | 141 |
| 105 | Definition of TPM2B_NAME Structure | 141 |
| 106 | Definition of TPMS_PCR_SELECT Structure | 142 |
| 107 | Definition of TPMS_PCR_SELECTION Structure | 142 |
| 108 | Values for proof Used in Tickets | 143 |
| 109 | General Format of a Ticket | 143 |
| 110 | Definition of TPMT_TK_CREATION Structure | 144 |
| 111 | Definition of TPMU_TK_VERIFIED_META Union | 145 |
| 112 | TPM_ST Values For Signature Verification Commands | 146 |
| 113 | Definition of TPMT_TK_VERIFIED Structure | 146 |
| 114 | Definition of TPMT_TK_AUTH Structure | 147 |
| 115 | Definition of TPMT_TK_HASHCHECK Structure | 148 |
| 116 | Definition of TPMS_ALG_PROPERTY Structure | 148 |
| 117 | Definition of TPMS_TAGGED_PROPERTY Structure | 149 |
| 118 | Definition of TPMS_TAGGED_PCR_SELECT Structure | 149 |
| 119 | Definition of TPMS_TAGGED_POLICY Structure | 149 |
| 120 | Definition of TPMS_ACT_DATA Structure | 149 |
| 121 | Definition of TPMS_SPDM_SESSION_INFO Structure | 150 |
| 122 | Definition of TPML_CC Structure | 150 |
| 123 | Definition of TPML_CCA Structure | 150 |
| 124 | Definition of TPML_ALG Structure | 151 |
| 125 | Definition of TPML_HANDLE Structure | 151 |
| 126 | Definition of TPML_DIGEST Structure | 151 |
| 127 | Definition of TPML_DIGEST_VALUES Structure | 152 |
| 128 | Definition of TPML_PCR_SELECTION Structure | 153 |
| 129 | Definition of TPML_ALG_PROPERTY Structure | 153 |
| 130 | Definition of TPML_TAGGED_TPM_PROPERTY Structure | 153 |
| 131 | Definition of TPML_TAGGED_PCR_PROPERTY Structure | 154 |
| 132 | Definition of (ECC) TPML_ECC_CURVE Structure | 154 |
| 133 | Definition of TPML_TAGGED_POLICY Structure | 154 |
| 134 | Definition of TPML_ACT_DATA Structure | 155 |
| 135 | Definition of TPML_PUB_KEY Structure | 155 |
| 136 | Definition of TPML_SPDM_SESSION_INFO Structure | 155 |
| 137 | Definition of TPML_VENDOR_PROPERTY Structure | 156 |
| 138 | Definition of TPMU_CAPABILITIES Union | 157 |
| 139 | Definition of TPMS_CAPABILITY_DATA Structure | 157 |
| 140 | Definition of TPMS_SET_CAPABILITY_DATA Structure | 157 |
| 141 | Definition of TPM2B_SET_CAPABILITY_DATA Structure | 158 |
| 142 | Definition of TPMS_CLOCK_INFO Structure | 158 |
| 143 | Definition of TPMS_TIME_INFO Structure | 159 |
| 144 | Definition of TPMS_TIME_ATTEST_INFO Structure | 160 |
| 145 | Definition of TPMS_CERTIFY_INFO Structure | 160 |
| 146 | Definition of TPMS_QUOTE_INFO Structure | 160 |
| 147 | Definition of TPMS_COMMAND_AUDIT_INFO Structure | 160 |
| 148 | Definition of TPMS_SESSION_AUDIT_INFO Structure | 161 |
| 149 | Definition of TPMS_CREATION_INFO Structure | 161 |
| 150 | Definition of TPMS_NV_CERTIFY_INFO Structure | 161 |
| 151 | Definition of TPMS_NV_DIGEST_CERTIFY_INFO Structure | 162 |
| 152 | Definition of (TPM_ST) TPML_ST_ATTEST Type | 162 |

| | | |
|-----|---|-----|
| 153 | Definition of TPMU_ATTEST Union | 162 |
| 154 | Definition of TPMS_ATTEST Structure | 163 |
| 155 | Definition of TPM2B_ATTEST Structure | 163 |
| 156 | Definition of TPMS_AUTH_COMMAND Structure | 164 |
| 157 | Definition of TPMS_AUTH_RESPONSE Structure | 164 |
| 158 | Definition of Types for Symmetric Cipher Key Sizes | 165 |
| 159 | Definition of TPMU_SYM_KEY_BITS Union | 165 |
| 160 | Definition of TPMU_SYM_MODE Union | 166 |
| 161 | Definition of TPMU_SYM_DETAILS Union | 167 |
| 162 | Definition of TPMT_SYM_DEF Structure | 167 |
| 163 | Definition of TPMT_SYM_DEF_OBJECT Structure | 167 |
| 164 | Definition of TPM2B_SYM_KEY Structure | 168 |
| 165 | Definition of TPMS_SYMCIPHER_PARMS Structure | 168 |
| 166 | Definition of TPM2B_LABEL Structure | 169 |
| 167 | Definition of TPMS_DERIVE Structure | 169 |
| 168 | Definition of TPM2B_DERIVE Structure | 169 |
| 169 | Definition of TPMU_SENSITIVE_CREATE Union | 169 |
| 170 | Definition of TPM2B_SENSITIVE_DATA Structure | 170 |
| 171 | Definition of TPMS_SENSITIVE_CREATE Structure | 170 |
| 172 | Definition of TPM2B_SENSITIVE_CREATE Structure | 170 |
| 173 | Definition of TPMS_SCHEME_HASH Structure | 171 |
| 174 | Definition of (ECC) TPMS_SCHEME_ECDAA Structure | 171 |
| 175 | Definition of (TPM_ALG_ID) TPMI_ALG_KEYEDHASH_SCHEME Type | 171 |
| 176 | Definition of Types for TPMS_SCHEME_HMAC | 172 |
| 177 | Definition of TPMS_SCHEME_XOR Structure | 172 |
| 178 | Definition of TPMU_SCHEME_KEYEDHASH Union | 172 |
| 179 | Definition of TPMT_KEYEDHASH_SCHEME Structure | 172 |
| 180 | Definition of (RSA) Types for RSA Signature Schemes | 173 |
| 181 | Definition of (ECC) Types for ECC Signature Schemes | 173 |
| 182 | Definition of TPMU_SIG_SCHEME Union | 174 |
| 183 | Definition of TPMT_SIG_SCHEME Structure | 174 |
| 184 | Definition of Types for RSA Encryption Schemes | 175 |
| 185 | Definition of Types for (ECC) ECC Key Exchange | 175 |
| 186 | Definition of Types for KDF Schemes | 175 |
| 187 | Definition of TPMU_KDF_SCHEME Union | 176 |
| 188 | Definition of TPMT_KDF_SCHEME Structure | 176 |
| 189 | Definition of (TPM_ALG_ID) (RSA) TPMI_ALG_RSA_SCHEME Type | 176 |
| 190 | Definition of (RSA) TPMU_RSA_SCHEME Union | 177 |
| 191 | Definition of (RSA) TPMT_RSA_SCHEME Structure | 177 |
| 192 | Definition of (TPM_ALG_ID) (RSA) TPMI_ALG_RSA_DECRYPT Type | 177 |
| 193 | Definition of (RSA) TPMT_RSA_DECRYPT Structure | 177 |
| 194 | Definition of (RSA) TPM2B_PUBLIC_KEY_RSA Structure | 178 |
| 195 | Definition of (RSA) (TPM_KEY_BITS) TPMI_RSA_KEY_BITS Type | 178 |
| 196 | Definition of (RSA) TPM2B_PRIVATE_KEY_RSA Structure | 178 |
| 197 | Definition of (ECC) TPM2B_ECC_PARAMETER Structure | 179 |
| 198 | Definition of (ECC) TPMS_ECC_POINT Structure | 179 |
| 199 | Definition of (ECC) TPM2B_ECC_POINT Structure | 179 |
| 200 | Definition of (TPM_ALG_ID) (ECC) TPMI_ALG_ECC_SCHEME Type | 180 |
| 201 | Definition of (ECC) (TPM_ECC_CURVE) TPMI_ECC_CURVE Type | 180 |
| 202 | Definition of (ECC) TPMU_ECC_SCHEME Union | 181 |
| 203 | Definition of (TPMT_SIG_SCHEME) (ECC) TPMT_ECC_SCHEME Structure | 181 |

| | | |
|-----|---|-----|
| 204 | Definition of (ECC) TPMS_ALGORITHM_DETAIL_ECC Structure | 182 |
| 205 | Definition of (MLKEM) TPMI_MLKEM_PARMS Type | 182 |
| 206 | Definition of (MLKEM) TPM2B_PUBLIC_KEY_MLKEM Structure | 183 |
| 207 | Definition of (MLKEM) TPM2B_PRIVATE_KEY_MLKEM Structure | 183 |
| 208 | Definition of (MLDSA) TPMI_MLDSA_PARMS Type | 184 |
| 209 | Definition of (PHMLDSA) TPMS_SIGNATURE_HASH_MLDSA Structure | 184 |
| 210 | Definition of (MLDSA) TPM2B_PUBLIC_KEY_MLDSA Structure | 184 |
| 211 | Definition of (MLDSA) TPM2B_PRIVATE_KEY_MLDSA Structure | 185 |
| 212 | Definition of (RSA) TPMS_SIGNATURE_RSA Structure | 185 |
| 213 | Definition of Types for (RSA) Signature | 185 |
| 214 | Definition of (ECC) TPMS_SIGNATURE_ECC Structure | 185 |
| 215 | Definition of Types for (ECC) TPMS_SIGNATURE_ECC | 185 |
| 216 | Definition of (ECC) TPM2B_SIGNATURE_EDDSA Structure | 186 |
| 217 | Definition of (MLDSA) TPM2B_SIGNATURE_MLDSA Structure | 186 |
| 218 | Definition of TPMU_SIGNATURE Union | 187 |
| 219 | Definition of TPMT_SIGNATURE Structure | 187 |
| 220 | Definition of TPMU_SIGNATURE_CTX Union | 188 |
| 221 | Definition of TPM2B_SIGNATURE_CTX Structure | 188 |
| 222 | Definition of TPM2B_SIGNATURE_HINT Structure | 189 |
| 223 | Definition of TPMU_ENCRYPTED_SECRET Union | 189 |
| 224 | Definition of TPM2B_ENCRYPTED_SECRET Structure | 190 |
| 225 | Definition of (TPM_ALG_ID) TPMI_ALG_PUBLIC Type | 191 |
| 226 | Definition of TPMU_PUBLIC_ID Union | 192 |
| 227 | Definition of TPMS_KEYEDHASH_PARMS Structure | 193 |
| 228 | Definition of (RSA) TPMS_RSA_PARMS Structure | 193 |
| 229 | Definition of (ECC) TPMS_ECC_PARMS Structure | 194 |
| 230 | Definition of (MLDSA) TPMS_MLDSA_PARMS Structure | 196 |
| 231 | Definition of (PHMLDSA) TPMS_HASH_MLDSA_PARMS Structure | 197 |
| 232 | Definition of (MLKEM) TPMS_MLKEM_PARMS Structure | 197 |
| 233 | Definition of TPMU_PUBLIC_PARMS Union | 198 |
| 234 | Definition of TPMT_PUBLIC_PARMS Structure | 198 |
| 235 | Definition of TPMT_PUBLIC Structure | 198 |
| 236 | Definition of TPM2B_PUBLIC Structure | 199 |
| 237 | Definition of TPM2B_TEMPLATE Structure | 200 |
| 238 | Definition of TPM2B_PRIVATE_VENDOR_SPECIFIC Structure | 201 |
| 239 | Definition of TPMU_SENSITIVE_COMPOSITE Union | 201 |
| 240 | Definition of TPMT_SENSITIVE Structure | 201 |
| 241 | Definition of TPM2B_SENSITIVE Structure | 202 |
| 242 | Definition of _PRIVATE Structure | 203 |
| 243 | Definition of TPM2B_PRIVATE Structure | 203 |
| 244 | Definition of TPMS_ID_OBJECT Structure | 203 |
| 245 | Definition of TPM2B_ID_OBJECT Structure | 204 |
| 246 | Definition of (UINT32) TPM_NV_INDEX Bits | 205 |
| 247 | Definition of TPM_NT Constants | 205 |
| 248 | Definition of TPMS_NV_PIN_COUNTER_PARAMETERS Structure | 206 |
| 249 | Definition of (UINT32) TPMA_NV Bits | 207 |
| 250 | Definition of (UINT64) TPMA_NV_EXP Bits | 211 |
| 251 | Definition of TPMS_NV_PUBLIC Structure | 211 |
| 252 | Definition of TPM2B_NV_PUBLIC Structure | 212 |
| 253 | Definition of TPMS_NV_PUBLIC_EXP_ATTR Structure | 213 |
| 254 | Definition of TPMU_NV_PUBLIC_2 Union | 213 |

| | | |
|-----|---|-----|
| 255 | Definition of TPMT_NV_PUBLIC_2 Structure | 214 |
| 256 | Definition of TPM2B_NV_PUBLIC_2 Structure | 214 |
| 257 | Definition of TPM2B_CONTEXT_SENSITIVE Structure | 215 |
| 258 | Definition of TPMS_CONTEXT_DATA Structure | 215 |
| 259 | Definition of TPM2B_CONTEXT_DATA Structure | 216 |
| 260 | Definition of TPMS_CONTEXT Structure | 217 |
| 261 | Definition of TPMS_CREATION_DATA Structure | 219 |
| 262 | Definition of TPM2B_CREATION_DATA Structure | 219 |
| 263 | Definition of (UINT32) TPM_AT Constants | 221 |
| 264 | Definition of (UINT32) TPM_AE Constants | 221 |
| 265 | Definition of TPMS_AC_OUTPUT Structure | 221 |
| 266 | Definition of TPML_AC_CAPABILITIES Structure | 222 |

1 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 the Reference Code.

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.

2 Terms and definitions

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

3 Symbols and Abbreviated Terms

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

4 Notation

4.1 Introduction

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

4.2 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 the below example names a collection of 16-bit constants and the following example shows equivalent C code.

Example:

Definition of (UINT16) COUNTING Constants

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

Example:

Code:

```
/* 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
```

4.3 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 the next example shows how typedefs are defined in this specification and the following example shows equivalent C-compatible code.

Example:

Definition of Types for Some Purpose

| Type | Name | Description |
|----------------|-----------|-------------|
| unsigned short | UINT16 | |
| UINT16 | SOME_TYPE | |
| unsigned long | UINT32 | |
| UINT32 | LAST_TYPE | |

Example:

Code:

```
/* 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;
```

4.4 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 the next example shows how an enumeration would be defined in this specification, along with equivalent C code.

Example:

Definition of (UINT16) CARD_SUIT Values

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

Example:

Code:

```
/* C language equivalent of the structure defined in the table above */  
typedef enum {  
    CLUBS      = 0x0000,  
    DIAMONDS   = 0x000D,  
    HEARTS     = 0x001A,  
    SPADES     = 0x0027  
} CARD_SUIT;
```

4.5 Interface Type

An interface type is used for an enumeration that is checked by the unmarshaling code. 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:

Definition of (CARD_SUIT) RED_SUIT Type

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

Example:

Code:

```

/* 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.

Example:

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

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 the example above, the tables in the example below, are equivalent.

Example:

Definition of (UINT16) MY_KEY_BITS Type

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

Definition of (UINT16) MY_KEY_BITS Type

| Parameter | Description |
|---------------------|---|
| \$MY_KEY_SIZES_BITS | the number of bits in the supported key |

4.6 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 the example below shows how a structure containing fixed and variable-length arrays would be defined in this specification, along with equivalent C code.

Example:

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:

Code:

```
/* C language equivalent of the typedefs from the table above */  
typedef struct {  
    UINT16 array1[20];  
    UINT16 a_size;  
    UINT32 array2[];  
} A_STRUCT;
```

4.7 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 the example below shows an example of how a structure would be defined in this specification and the following example shows equivalent C code.

Example:

Definition of SIMPLE_STRUCTURE Structure

| Parameter | Type | Description |
|-----------|--------|-------------|
| tag | TPM_ST | |
| value1 | INT32 | |
| value2 | INT32 | |

Example:

Code:

```
/* C language equivalent of the structure defined in the table above */
typedef struct {
    TPM_ST tag;
    INT32 value1;
    INT32 value2;
} SIMPLE_STRUCTURE;
```

Example:

Code:

```
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 parentheses (()). 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 192 - Definition of (TPM_ALG_ID) (RSA) TPMI_ALG_RSA_DECRYPT Type, which is dependent on the RSA algorithm.

4.8 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 have an appended "+", then the conditional type is not allowed.

Example:

Table 77 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:

Table 163 - 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_RSA_PARMS. The "+" in TPMS_RSA_PARMS means that the algorithm parameter in the TPMT_SYM_DEF_OBJECT may have the conditional value (TPM_ALG_NULL).

Example:

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:

Definition of POKER_CARD Structure

| Parameter | Type | Description |
|-----------|-----------------|----------------|
| suit | TPMI_CARD_SUIT+ | allows joker |
| number | UINT8 | the card value |

Example:

Definition of BRIDGE_CARD Structure

| Parameter | Type | Description |
|-----------|----------------|----------------------|
| suit | TPMI_CARD_SUIT | does not allow joker |
| number | UINT8 | the card value |

4.9 Unions

4.9.1 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.

4.9.2 Union Definition

The table in the example below 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 the example below illustrates a union in which a conditional null selector is allowed to indicate an empty union member.

The following example 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:

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:

Code:

```
// 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:

Code:

```
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 sub-classes. Since there is no selector, this union member cannot be marshaled or unmarshaled.

Example:

Table 218 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.

4.9.3 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 the below example shows the definition of a structure that contains a union and a selector. The following example shows equivalent C-compatible code and the example after that shows how the unmarshaling code would handle the selector.

Example:

Definition of STRUCTURE_WITH_UNION Structure

| Parameter | Type | Description |
|-----------------|---------------|--|
| select | NUMBER_SELECT | a value indicating the type in <i>number</i> |
| [select] number | NUMBER_UNION | a union as shown in Union Definition |

Example:

Code:

```
// C-compatible version of the union structure in the table above
typedef struct {
    NUMBER_SELECT select;
    NUMBER_UNION number;
} STRUCT_WITH_UNION;
```

Example:

Code:

```
// 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));
}
```

4.9.4 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 Clause 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 the next example, the table defines a value for each of the union members.

Example:

Definition of (INT8) NUMBER_SELECT Values

| 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 Clause 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 the next example 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_INTEGER 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:

Definition of (NUMBER_SELECT) AN_INTEGER Type

| 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 the next example.

The selector in the previous example can then be used in a subset union as shown in the next example:

Example:

Definition of JUST_INTEGERS Structure

| Parameter | Type | Description |
|-----------------|--------------|--|
| select | AN_INTEGER | a value indicating the type in <i>number</i> |
| [select] number | NUMBER_UNION | a union as shown in Union Definition |

4.10 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 the below example shows how a structure containing bit fields would be defined in this specification, along with equivalent C code.

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.

Example:

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:

Code:

```
/* 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.

4.11 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:

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 <i>value1</i> , 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. |
| 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 <i>value4</i> shall have a value of 25 or less. From the definition of <i>value4</i> above, it can determine that <i>value4</i> must have a value of at least 20. |

4.12 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 the example below shows a structure where the size parameter would nominally indicate the number of octets in the remainder of the structure.

Example:

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 validation code 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:

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 <i>size</i> . That is, the number of bytes in the input buffer used to successfully unmarshal <i>someStructure</i> must be the same as <i>size</i> . |
| 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 <i>size</i> , the TPM is required to validate that the unmarshaled size exactly matches <i>size</i> . |
| someData | UINT32 | a value |

4.13 Name Prefix Convention

Table 1 describes the parameters, 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 one of: TPM_ST_, TPM_ALG_, or TPM_HT_, 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. |

(continued on next page)

(continued from previous page)

| Prefix | Description |
|---------|---|
| TPM_xx_ | <p>an enumeration value of a particular type</p> <p>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.</p> <p>Example:</p> <p>TPM_CC_ indicates that the type is used for a <i>commandCode</i>. The allowed enumeration values will be found in the table defining the TPM_CC constants (see Table 12)</p> <p>Example:</p> <p>TPM_RC_ indicates that the type is used for a <i>responseCode</i>. The allowed enumeration values are in Table 18.</p> |

4.14 Data Alignment

The data structures in this specification 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.

4.15 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:

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:

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.

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.

5 Base Types

5.1 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 are normally 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 <code>int</code> 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 |

5.2 Specification Logic Value Constants

Table 4 defines logical values.

Table 4: Defines for Logic Values

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

(continued on next page)

(continued from previous page)

| Name | Value | Description |
|-------|-------|-------------|
| SET | 1 | |
| CLEAR | 0 | |

5.3 Miscellaneous Types

5.3.1 Key Size Types

Table 5 defines types that are used to express the sizes of keys.

Table 5: Definition of Types for TPM_KEY_BITS

| Type | Name | Description |
|--------|--------------|--------------------|
| UINT16 | TPM_KEY_BITS | a key size in bits |

6 Constants

6.1 TPM_SPEC (Specification Version Values)

Table 6 defines values that are readable with TPM2_GetCapability() (see Clause 6.12.1 for the format).

Note:

This table will change with each published version of the specification. However, an implementation of a Library specification that addresses published errata must use the TPM_SPEC table stated in the errata (to indicate its compatibility with the errata), not the TPM_SPEC table in the Library specification.

Note:

Prior to version 184, the TPM 2.0 specification's version number was equal to TPM_SPEC_VERSION divided by 100, e.g., version 1.83 (published March 2024).

Beginning with version 184, the TPM 2.0 specification's version number is equal to TPM_SPEC_VERSION.

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 | 185 | the version number of the specification |
| TPM_SPEC_YEAR | 0 | shall be zero Note: Prior to version 185, this constant reported the year in which the TPM Library Specification indicated by TPM_SPEC_VERSION (or an Errata) was published. It is set to 0 to indicate that the errata level is reported in TPM_SPEC_ERRATA. |
| TPM_SPEC_ERRATA | 0 | the errata version implemented by the TPM This number will increment for every errata document published for this TPM_SPEC_VERSION Note: Prior to version 185, this constant was called TPM_SPEC_DAY_OF_YEAR. |

6.2 TPM_CONSTANTS32

Table 7 defines 32-bit, miscellaneous constant values used by the TPM.

Note:

Currently, there are no other sizes for miscellaneous constants.

Table 7: Definition of (UINT32) TPM_CONSTANTS32 Constants

| Name | Value | Comments |
|-------------------------|------------|---|
| TPM_GENERATED_VALUE | 0xff544347 | 0xFF 'TCG' (FF 54 43 47) value used to differentiate TPM-generated structures from non-TPM structures |
| TPM_MAX_DERIVATION_BITS | 8192 | the maximum number of bits that may be generated by an instantiation of the deterministic pseudo-random bit generator |

6.3 TPM_ALG_ID

Table 8 is a list of TPM-relevant algorithms, a proper subset of the TCG Algorithm Registry [1]. Each value's name is aliased with the prefix TPM_ALG instead of TCG_ALG. The numeric values listed in Table 8 are the same as the numeric values listed in the TCG Algorithm Registry.

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.

Note:

Table 8 lists only algorithms that are currently supported by the TPM 2.0 Library.

Some algorithm IDs in the TCG Algorithm Registry are not yet supported by TPMs, but are intended for consideration in a future version of the TPM 2.0 Library.

An algorithm ID is often used like a tag to determine the type of a structure in a context-sensitive way.

TPM_ALG_KEYEDHASH and TPM_ALG_NULL are required of all TPM implementations.

Table 8: Definition of (UINT16) TPM_ALG_ID Constants

| Algorithm Name | TCG_ALG Value | Reference | Comments |
|---------------------|---------------|-----------------------------------|--|
| Object Types | | | |
| TPM_ALG_KEYEDHASH | 0x0008 | 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_SYMCIPHER | 0x0025 | TCG TPM 2.0 library specification | the object type for a symmetric block cipher |

(continued on next page)

(continued from previous page)

| Algorithm Name | TCG_ALG Value | Reference | Comments |
|---------------------------|---------------|--------------------------|--|
| TPM_ALG_RSA | 0x0001 | IETF RFC 8017 [2] | the RSA algorithm |
| TPM_ALG_ECC | 0x0023 | ISO/IEC 15946-1 [3] | prime field ECC |
| TPM_ALG_MLKEM | 0x00A0 | NIST FIPS 203 [4] | ML-KEM |
| TPM_ALG_MLDSA | 0x00A1 | NIST FIPS 204 [5] | ML-DSA |
| TPM_ALG_HASH_MLDSA | 0x00A2 | NIST FIPS 204 [5] | HashML-DSA |
| Hash Algorithms | | | |
| TPM_ALG_SHA1 | 0x0004 | ISO/IEC 10118-3 [6] | Deprecated. See Part 0. |
| TPM_ALG_SHA256 | 0x000B | ISO/IEC 10118-3 [6] | the SHA 256 algorithm |
| TPM_ALG_SHA384 | 0x000C | ISO/IEC 10118-3 [6] | the SHA 384 algorithm |
| TPM_ALG_SHA512 | 0x000D | ISO/IEC 10118-3 [6] | the SHA 512 algorithm |
| TPM_ALG_SM3_256 | 0x0012 | ISO/IEC 10118-3:2018 [6] | SM3 hash algorithm |
| TPM_ALG_SHA3_256 | 0x0027 | ISO/IEC 10118-3 [6] | Hash algorithm producing a 256 bit digest |
| TPM_ALG_SHA3_384 | 0x0028 | ISO/IEC 10118-3 [6] | Hash algorithm producing a 384 bit digest |
| TPM_ALG_SHA3_512 | 0x0029 | ISO/IEC 10118-3 [6] | Hash algorithm producing a 512 bit digest |
| Block Ciphers | | | |
| TPM_ALG_TDES | 0x0003 | ISO/IEC 18033-3 [7] | Deprecated. See Part 0. |
| TPM_ALG_AES | 0x0006 | ISO/IEC 18033-3 [7] | the AES algorithm with various key sizes |
| TPM_ALG_SM4 | 0x0013 | GB/T 32907-2016 [8] | SM4 symmetric block cipher |
| TPM_ALG_CAMELLIA | 0x0026 | ISO/IEC 18033-3 [7] | Camellia is a symmetric block cipher. The Camellia algorithm has various key sizes. |
| Block Cipher Modes | | | |
| TPM_ALG_CTR | 0x0040 | ISO/IEC 10116 [9] | Counter mode - if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode. |
| TPM_ALG_OFB | 0x0041 | ISO/IEC 10116 [9] | Output Feedback mode - if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode. |

(continued on next page)

(continued from previous page)

| Algorithm Name | TCG_ALG Value | Reference | Comments |
|-------------------------------------|---------------|--------------------------|---|
| TPM_ALG_CBC | 0x0042 | ISO/IEC 10116 [9] | Cipher Block Chaining mode - if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode. |
| TPM_ALG_CFB | 0x0043 | ISO/IEC 10116 [9] | Cipher Feedback mode - if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode. |
| TPM_ALG_ECB | 0x0044 | ISO/IEC 10116 [9] | Electronic Codebook mode - if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode. Note: This mode is not recommended unless the key is frequently rotated, such as in video codecs. |
| Message Authentication Codes | | | |
| TPM_ALG_HMAC | 0x0005 | ISO/IEC 9797-2 [10] | Hash Message Authentication Code (HMAC) algorithm |
| TPM_ALG_CMAC | 0x003F | ISO/IEC 9797-1:2011 [10] | Block Cipher-based Message Authentication Code (CMAC) "Algorithm 5" in ISO/IEC 9797-1:2011 |
| Key Derivation Functions | | | |
| TPM_ALG_HKDF | 0x001F | IETF RFC 5869 [11] | HMAC-based Extract-and-Expand Key Derivation Function |

(continued on next page)

(continued from previous page)

| Algorithm Name | TCG_ALG Value | Reference | Comments |
|------------------------|---------------|--|--|
| TPM_ALG_KDF1_SP800_56A | 0x0020 | NIST SP 800-56A [12] NIST SP 800-56C [13] | a key derivation method “(Cofactor) One-Pass Diffie-Hellman, C(1e, 1s, ECC CDH)” from SP 800-56A, using the concatenation format for <i>FixedInfo</i> . The key derivation method (KDM) used is the “One-Step Key Derivation” from SP 800-56C, where the Auxiliary Function is HMAC (Option 2). |
| TPM_ALG_KDF1_SP800_108 | 0x0022 | NIST SP 800-108 [14] | a key derivation method clause 5.1 KDF in Counter Mode |
| TPM_ALG_KDF2 | 0x0021 | IEEE Std 1363a-2004 [15] | key derivation function KDF2 clause 13.2 |
| RSA Schemes | | | |
| TPM_ALG_RSASSA | 0x0014 | IETF RFC 8017 [2] | a signature algorithm defined in clause 8.2 (RSASSA-PKCS1-v1_5) |
| TPM_ALG_RSAPSS | 0x0016 | IETF RFC 8017 [2] | a signature algorithm defined in clause 8.1 (RSASSA-PSS) |
| TPM_ALG_RSAES | 0x0015 | IETF RFC 8017 [2] | a padding algorithm defined in clause 7.2 (RSAES-PKCS1-v1_5) |
| TPM_ALG_OAEP | 0x0017 | IETF RFC 8017 [2] | a padding algorithm defined in clause 7.1 (RSAES_OAEP) |
| ECC Schemes | | | |
| TPM_ALG_ECDSA | 0x0018 | ISO/IEC 14888-3 [16] | signature algorithm using elliptic curve cryptography (ECC) (Non-deterministic ECDSA) |
| TPM_ALG_EC Schnorr | 0x001C | TCG TPM 2.0 library specification | elliptic-curve based Schnorr signature |
| TPM_ALG_ECDA | 0x001A | TCG TPM 2.0 library specification | elliptic-curve based, anonymous signing scheme |

(continued on next page)

(continued from previous page)

| Algorithm Name | TCG_ALG Value | Reference | Comments |
|----------------------|---------------|--|---|
| TPM_ALG_ECDH | 0x0019 | NIST SP 800-56A [12] IETF RFC 7748 [17] | Elliptic Curve Diffie-Hellman Based on context, this can be one of: <ul style="list-style-type: none">• “Elliptic Curve Cryptography Cofactor Diffie-Hellman (ECC CDH) Primitive” defined in 5.7.1.2 of SP 800-56A• X25519 or X448 as defined in clause 5 of RFC 7748 |
| TPM_ALG_ECMQV | 0x001D | NIST SP 800-56A [12] | “Full MQV, C(2e, 2s, ECC MQV)” defined in 6.1.1.4 |
| TPM_ALG_SM2 | 0x001B | GB/T 32918.1-2016 [18] GB/T 32918.2-2016 [19] GB/T 32918.3-2016 [20] GB/T 32918.4-2016 [21] GB/T 32918.5-2017 [22] | SM2 - depending on context, either an elliptic-curve based signature algorithm, an encryption scheme or a key exchange protocol |
| TPM_ALG_EDDSA | 0x0060 | RFC 8032 [23] | Edwards-curve Digital Signature Algorithm (PureEdDSA) |
| TPM_ALG_HASH_EDDSA | 0x0061 | RFC 8032 [23] | Edwards-curve Digital Signature Algorithm (HashEdDSA) Note: Earlier TCG specifications used the name TPM_ALG_EDDSA_PH. |
| Miscellaneous | | | |
| TPM_ALG_NULL | 0x0010 | TCG TPM 2.0 library specification | Null algorithm |
| TPM_ALG_XOR | 0x000A | TCG TPM 2.0 library specification | the XOR encryption algorithm |
| TPM_ALG_MGF1 | 0x0007 | IEEE Std 1363™-2000 [24] IEEE Std 1363a™-2004 [15] | hash-based mask-generation function |

(continued on next page)

(continued from previous page)

| Algorithm Name | TCG_ALG Value | Reference | Comments |
|-----------------|-----------------------|-----------|----------|
| <i>Reserved</i> | | | |
| reserved | 0x0000 | | reserved |
| reserved | 0x00C1 through 0x00C6 | | reserved |
| reserved | 0x8000 through 0xFFFF | | reserved |

6.4 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 9 is a copy of the TPM_ECC_CURVE constants table in the TCG registry as of the date of publication of this specification. Table 9 is provided for illustrative purposes only.

Table 9: Definition of (UINT16) (ECC) TPM_ECC_CURVE Constants

| 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 ECDA |
| TPM_ECC_BN_P638 | 0x0011 | curve to support ECDA |
| TPM_ECC_SM2_P256 | 0x0020 | |
| TPM_ECC_BP_P256_R1 | 0x0030 | Brainpool |
| TPM_ECC_BP_P384_R1 | 0x0031 | Brainpool |
| TPM_ECC_BP_P512_R1 | 0x0032 | Brainpool |
| TPM_ECC_CURVE_25519 | 0x0040 | curve to support EdDSA |
| TPM_ECC_CURVE_448 | 0x0041 | curve to support EdDSA |

6.5 TPM_CC

6.5.1 Format

A command code is a 32-bit structure with fields assigned as shown in Table 10. Table 11 defines the bits for these fields.

Note:

Responses from TPM2_GetCapability(*capability* == TPM_CAP_COMMANDS) use the related TPMA_CC type, which has the same bit assignments except certain TPM_CC reserved bits are allowed to be set.

Table 10: TPM Command (TPM_CC) Format

| | | | | | | | | | | | | | | | | |
|----------|----|----|----------------------------|----|----|----|----|----------|----|----|--|----|---------------|--|----|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | | 16 | 15 | | 00 | |
| Reserved | | V | Reserved (used in TPMA_CC) | | | | | Reserved | | | | | Command Index | | | |

Table 11: TPM Command (TPM_CC) Fields Description

| Bit | Name | Definition |
|-----------|---------------|---|
| 15:0 (16) | Command Index | the index of the command |
| 21:16 (6) | Reserved | shall be zero |
| 28:22 (7) | Reserved | shall be zero Note: In TPMA_CC, these bits are allowed to be set. |
| 29 (1) | V | Vendor Indicator SET (1): The command is vendor-specific CLEAR (0): The command is defined by the TCG in Table 12 |
| 31:30 (2) | Reserved | shall be zero |

6.5.2 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)

| 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 | | |

(continued on next page)

(continued from previous page)

| Name | Command Code | Dep | Comments |
|-----------------------------------|--------------|-----|----------|
| 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 |

(continued on next page)

(continued from previous page)

| Name | Command Code | Dep | Comments |
|----------------------------------|--------------|-------|----------------|
| 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 below |
| TPM_CC_MAC | 0x00000155 | CMAC | see NOTE below |
| TPM_CC_Import | 0x00000156 | | |
| TPM_CC_Load | 0x00000157 | | |

(continued on next page)

(continued from previous page)

| Name | Command Code | Dep | Comments |
|---------------------------|--------------|-------|--------------------------------|
| TPM_CC_Quote | 0x00000158 | | |
| TPM_CC_RSA_Decrypt | 0x00000159 | RSA | |
| TPM_CC_HMAC_Start | 0x0000015B | !CMAC | see NOTE below |
| TPM_CC_MAC_Start | 0x0000015B | CMAC | see NOTE below |
| TPM_CC_SequenceUpdate | 0x0000015C | | |
| TPM_CC_Sign | 0x0000015D | | Deprecated. See Part 0. |
| 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 | | Deprecated. See Part 0. |
| 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 | |

(continued on next page)

(continued from previous page)

| Name | Command Code | Dep | Comments |
|--------------------------------|--------------|-----|--------------------------------|
| TPM_CC_StartAuthSession | 0x00000176 | | |
| TPM_CC_VerifySignature | 0x00000177 | | Deprecated. See Part 0. |
| 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 |

(continued on next page)

(continued from previous page)

| Name | Command Code | Dep | Comments |
|-------------------------------|--------------|-----|--------------------------------|
| TPM_CC_CreateLoaded | 0x00000191 | | Deprecated. See Part 0. |
| TPM_CC_PolicyAuthorizeNV | 0x00000192 | | Policy |
| TPM_CC_EncryptDecrypt2 | 0x00000193 | | |
| TPM_CC_AC_GetCapability | 0x00000194 | | Deprecated. See Part 0. |
| TPM_CC_AC_Send | 0x00000195 | | Deprecated. See Part 0. |
| TPM_CC_Policy_AC_SendSelect | 0x00000196 | | Deprecated. See Part 0. |
| TPM_CC_CertifyX509 | 0x00000197 | | Deprecated. See Part 0. |
| TPM_CC_ACT_SetTimeout | 0x00000198 | | |
| TPM_CC_ECC_Encrypt | 0x00000199 | ECC | |
| TPM_CC_ECC_Decrypt | 0x0000019A | ECC | |
| TPM_CC_PolicyCapability | 0x0000019B | | Policy |
| TPM_CC_PolicyParameters | 0x0000019C | | Policy |
| TPM_CC_NV_DefineSpace2 | 0x0000019D | | |
| TPM_CC_NV_ReadPublic2 | 0x0000019E | | |
| TPM_CC_SetCapability | 0x0000019F | | |
| TPM_CC_ReadOnlyControl | 0x000001A0 | | |
| TPM_CC_PolicyTransportSPDM | 0x000001A1 | | |
| Reserved | 0x000001A2 | | |
| TPM_CC_VerifySequenceComplete | 0x000001A3 | | |
| TPM_CC_SignSequenceComplete | 0x000001A4 | | |
| TPM_CC_VerifyDigestSignature | 0x000001A5 | | |
| TPM_CC_SignDigest | 0x000001A6 | | |
| TPM_CC_Encapsulate | 0x000001A7 | | |
| TPM_CC_Decapsulate | 0x000001A8 | | |
| TPM_CC_VerifySequenceStart | 0x000001A9 | | |
| TPM_CC_SignSequenceStart | 0x000001AA | | |

(continued on next page)

(continued from previous page)

| Name | Command Code | Dep | Comments |
|------------------------|----------------|-----|--|
| TPM_CC_LAST | 0x000001AA | | 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:

A TPM can 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.

6.6 TPM_RC

6.6.1 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 00 00 0₁₆ and the low-order 12 bits (11:0) 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.

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.

6.6.2 Response Code Formats

The response codes for this specification are defined in Table 14, Table 16, and Table 17 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. The first group (format-zero) contains the response codes for this specification that are not related to command parameters. The second group (format-one) contains the errors that may be associated with a command parameter, handle, or session.

Table 13 shows the format for the response codes when bit 7 is zero.

Table 13: Format-Zero Response Codes (TPM_RC)

| | | | | | | | | | | | | | | |
|-----------------------|--|----|----|----|------|---|---|---|---|---|---|---|---|---|
| 31 | | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 00 00 0 ₁₆ | | | S | T | Res. | V | 0 | E | | | | | | |

The field definitions are:

Table 14: Format-Zero Response Codes (TPM_RC) Fields Description

| Bit | Name | Definition |
|---------|----------|---|
| 6:0 (7) | E | Error Number The interpretation of this field is dependent on the setting of the F and S fields. |
| 7 (1) | F | Format Selector CLEAR (0) for the format in this table or when the response code is TPM_RC_BAD_TAG. |
| 8 (1) | 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. |
| 9 (1) | Reserved | shall be zero |
| 10 (1) | 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). <i>Note:</i> This attribute does not indicate a vendor-specific code unless the F attribute (bit 7) is CLEAR. |

(continued on next page)

(continued from previous page)

| Bit | Name | Definition |
|--------|------|--|
| 11 (1) | 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. Table 15 shows the format for the response codes when bit 7 is one.

Table 15: Format-One Response Codes (TPM_RC)

| 31 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-----------------------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 00 00 0 ₁₆ | | N | | | | 1 | P | E | | | | | |

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 the low-order 6 bits (5:0). When an error is associated with a parameter, TPM_RC_P (0x040) is added and N is set to the parameter number.

Note:

In the Reference Code, 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 (0x040) 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:

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 16: Format-One Response Codes (TPM_RC) Fields Description

| Bit | Name | Definition |
|-----|------|------------|
|-----|------|------------|

(continued on next page)

(continued from previous page)

| Bit | Name | Definition |
|----------|------|--|
| 5:0 (6) | E | Error Number The error number is independent of the other settings. |
| 6 (1) | P | Parameter Indicator SET (1): The error is associated with a parameter. CLEAR (0): The error is associated with a handle or a session. |
| 7 (1) | F | Format Selector SET (1) for the format in this table. |
| 11:8 (4) | N | 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 0b0000 and 0b0111. Sessions use values between 0b1000 and 0b1111. Note: Bit 11 distinguishes between handles and sessions. Bits 10:8 all set to zero indicates that the number is unspecified. |

The groupings of response codes are determined by bits 8, 7, and 6 of the response code as summarized in Table 17.

Table 17: Response Code Groupings

| Bit | | | Definition |
|-----|---|---|--|
| 8 | 7 | 6 | |
| 0 | 0 | x | a response code not defined by this specification 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 |

6.6.3 TPM_RC Values

In general, response codes defined in TPM 2.0 Part 2 Table 18 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 18: Definition of (UINT32) TPM_RC Constants (Actions)

| Name | Value | Description |
|---------------------|-----------------|--|
| TPM_RC_SUCCESS | 0x000 | |
| TPM_RC_BAD_TAG | 0x01E | |
| Format-Zero | | |
| 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 can be returned by TPM2_GetTestResult() as the <i>testResult</i> 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. |

(continued on next page)

(continued from previous page)

| Name | Value | Description |
|--------------------------|-----------------|--|
| 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 <i>commandSize</i> 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 <i>authorizationSize</i> 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 (written) 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 |

(continued on next page)

(continued from previous page)

| Name | Value | Description |
|-------------------|-----------------|--|
| 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 |
| TPM_RC_READ_ONLY | RC_VER1 + 0x056 | command failed because the TPM is in the Read-Only mode of operation. |
| RC_MAX_FM0 | RC_VER1 + 0x07F | largest version 1 code that is not a warning |
| Format-One | | |
| 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. |

(continued on next page)

(continued from previous page)

| Name | Value | Description |
|----------------------|-----------------|--|
| TPM_RC_AUTH_FAIL | RC_FMT1 + 0x00E | the authorization HMAC check failed and the DA counter was incremented, or use of lockoutAuth is disabled |
| 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 <i>commandCode</i> in the policy is not the <i>commandCode</i> 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. |
| TPM_RC_FW_LIMITED | RC_FMT1 + 0x028 | the hierarchy is firmware-limited but the Firmware Secret is unavailable |
| TPM_RC_SVN_LIMITED | RC_FMT1 + 0x029 | the hierarchy is SVN-limited but the Firmware SVN Secret associated with the given SVN is unavailable |
| TPM_RC_PARMS | RC_FMT1 + 0x02A | parameter set not supported |
| TPM_RC_EXT_MU | RC_FMT1 + 0x02B | external-Mu is not supported |

(continued on next page)

(continued from previous page)

| Name | Value | Description |
|---------------------------|-----------------|--|
| TPM_RC_ONE_SHOT_SIGNATURE | RC_FMT1 + 0x02C | the TPM does not support signing arbitrarily long messages using this key. The entire message needs to be provided in the <i>buffer</i> parameter of TPM2_SignSequenceComplete(). |
| TPM_RC_SIGN_CONTEXT_KEY | RC_FMT1 + 0x02D | the key being used to finish the signature context is not the same as the one that was used to start it. |
| TPM_RC_CHANNEL | RC_FMT1 + 0x030 | command requires secure channel protection |
| TPM_RC_CHANNEL_KEY | RC_FMT1 + 0x031 | secure channel was not established with required requester or TPM key |
| Warnings | | |
| 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: This cannot occur when using the Reference Code. Note: 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 |

(continued on next page)

(continued from previous page)

| Name | Value | Description |
|---------------------|-----------------|--|
| 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 when using the Reference Code. |
| 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 |

(continued on next page)

(continued from previous page)

| Name | Value | Description |
|---------------------------|-----------------|--|
| TPM_RC_NV_RATE | RC_WARN + 0x020 | the TPM is rate-limiting accesses to prevent wear out 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 |

6.7 TPM_CLOCK_ADJUST

A TPM_CLOCK_ADJUST value in Table 19 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 coarse 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 19: Definition of (INT8) TPM_CLOCK_ADJUST Constants

| Name | Value | Comments |
|-------------------------|-------|--|
| TPM_CLOCK_COARSE_SLOWER | -3 | slow the <i>Clock</i> update rate by one coarse adjustment step |
| TPM_CLOCK_MEDIUM_SLOWER | -2 | slow the <i>Clock</i> update rate by one medium adjustment step |
| TPM_CLOCK_FINE_SLOWER | -1 | slow the <i>Clock</i> update rate by one fine adjustment step |
| TPM_CLOCK_NO_CHANGE | 0 | no change to the <i>Clock</i> update rate |
| TPM_CLOCK_FINE_FASTER | 1 | speed the <i>Clock</i> update rate by one fine adjustment step |
| TPM_CLOCK_MEDIUM_FASTER | 2 | speed the <i>Clock</i> update rate by one medium adjustment step |
| TPM_CLOCK_COARSE_FASTER | 3 | speed the <i>Clock</i> update rate by one coarse adjustment step |
| #TPM_RC_VALUE | | |

6.8 TPM_EO

Table 20 defines the TPM_EO constants for the EA arithmetic operands.

The signed arithmetic operations are performed using two's complement.

Note:

For two negative values, TPMs prior to version 1.83 might have implemented the signed arithmetic operations using sign-magnitude.

Table 20: Definition of (UINT16) TPM_EO Constants

| Operation Name | Value | Comments |
|------------------|--------|--------------|
| TPM_EO_EQ | 0x0000 | A = B |
| TPM_EO_NEQ | 0x0001 | A ≠ B |
| TPM_EO_SIGNED_GT | 0x0002 | A > B signed |

(continued on next page)

(continued from previous page)

| Operation Name | Value | Comments |
|--------------------|--------|---|
| 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 |

6.9 TPM_ST

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 families 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 21 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 21: Definition of (UINT16) TPM_ST Constants

| Name | Value | Comments |
|------|-------|----------|
|------|-------|----------|

(continued on next page)

(continued from previous page)

| Name | Value | Comments |
|--------------------|--------|--|
| TPM_ST_RSP_COMMAND | 0x00C4 | <p><i>tag</i> 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 (0x001E), which has the same numeric value as the TPM 1.2 response code for TPM_BADTAG.</p> <p>Note:</p> <p>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 may return the old value instead of the new value.</p> |
| TPM_ST_NULL | 0X8000 | no structure type specified |
| TPM_ST_NO_SESSIONS | 0x8001 | <p><i>tag</i> value for a command/response for a command defined in this specification; indicating that the command/response has no attached sessions and no <i>authorizationSize/parameterSize</i> value is present</p> <p>If the <i>responseCode</i> from the TPM is not TPM_RC_SUCCESS, then the response tag shall have this value.</p> |
| TPM_ST_SESSIONS | 0x8002 | <p><i>tag</i> 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 <i>authorizationSize/parameterSize</i> field is present</p> |

(continued on next page)

(continued from previous page)

| Name | Value | Comments |
|-----------------------------|--------|---|
| reserved | 0x8003 | <p>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.</p> <p>Note:</p> <p>The response to application software will have a <i>tag</i> of TPM_ST_NO_SESSIONS.</p> <p>Between the TRM and TPM, this tag would occur in a response from a TPM that overlaps the <i>tag</i> parameter of a request with the <i>tag</i> parameter of a response, when the response has no associated sessions.</p> <p>Note:</p> <p>This tag is not used by all TPM or TRM implementations.</p> |
| reserved | 0x8004 | <p>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.</p> <p>Note:</p> <p>If the command completes successfully, the response to application software will have a <i>tag</i> of TPM_ST_SESSIONS.</p> <p>Between the TRM and TPM, would occur in a response from a TPM that overlaps the <i>tag</i> parameter of a request with the <i>tag</i> parameter of a response, when the response has authorization sessions.</p> <p>Note:</p> <p>This tag is not used by all TPM or TRM implementations.</p> |
| 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 |

(continued on next page)

(continued from previous page)

| Name | Value | Comments |
|-------------------------|--------|---|
| 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 TPMT_TK_VERIFIED produced by TPM2_VerifySignature() |
| 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_MESSAGE_VERIFIED | 0x8026 | tag for a TPMT_TK_VERIFIED produced by TPM2_VerifySequenceComplete() |
| TPM_ST_DIGEST_VERIFIED | 0x8027 | tag for a TPMT_TK_VERIFIED produced by TPM2_VerifyDigestSignature() |
| TPM_ST_FU_MANIFEST | 0x8029 | tag for a structure describing a Field Upgrade Policy |

6.10 TPM_SU

The Table 22 startup type 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 that the startup type is TPM_SU_NONE so the reserved value is defined in the implementation as required for internal TPM uses.

Table 22: Definition of (UINT16) TPM_SU Constants

| 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 |

6.11 TPM_SE

The Table 23 session types are used in TPM2_StartAuthSession() to indicate the type of the session to be created.

Table 23: Definition of (UINT8) TPM_SE Constants

| Name | Value | Description |
|---------------|-------|--|
| TPM_SE_HMAC | 0x00 | |
| TPM_SE_POLICY | 0x01 | |
| TPM_SE_TRIAL | 0x03 | the policy session is being used to compute the <i>policyHash</i> 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 |

6.12 TPM_CAP

The Table 24 capability 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 24: 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 |

(continued on next page)

(continued from previous page)

| Capability Name | Value | Property Type | Return Type |
|---------------------------|------------|--------------------------|------------------------------|
| 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_PCERS | 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_PUB_KEYS | 0x0000000B | TPM_PUB_KEY (5) | TPML_PUB_KEY |
| TPM_CAP_SPDM_SESSION_INFO | 0x0000000C | reserved (5) | TPML_SPDM_SESSION_INFO |
| TPM_CAP_LAST | 0x0000000C | | |
| TPM_CAP_VENDOR_PROPERTY | 0x00000100 | manufacturer specific | manufacturer-specific values |
| #TPM_RC_VALUE | | | |

Note:

- [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 version 1.38.
- [4] TPM_CAP_ACT was added in version 1.59.
- [5] TPM_CAP_PUB_KEYS and TPM_CAP_SPDM_SESSION_INFO were added in version 184.

6.12.1 Settable Capabilities

There are two types of capabilities, which are identified by the capability value:

- Read-only capabilities, which can be read with TPM2_GetCapability()
- Settable capabilities, which can be set with TPM2_SetCapability() and read with TPM2_GetCapability().

Settable capabilities start at value 0x80000000.

Note:

Settable capabilities were added in version 1.83.

A capability is a 32-bit structure with fields assigned as shown in Table 25. Table 26 defines the bits.

Table 25: TPM Capability (TPM_CAP) Format

| | | | | | | | | | | | | | | |
|----|----------|----|----|----|----|----|----|----|----|--|----|----|--|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | | 16 | 15 | | 00 |
| S | Reserved | | | A | | | V | W | | | C | | | |

Table 26: TPM Capability (TPM_CAP) Fields Description

| Bit | Name | Definition |
|-----------|----------|--|
| 15:0 (16) | C | Capability |
| 22:16 (7) | W | Vendor Identifier Shall be zero if the capability is non-settable or if the capability is settable but defined by the TCG. For settable, vendor-specific capabilities, identifies the vendor Note: See TCG TPM Vendor ID Registry Family 1.2 and 2.0 for a list of Vendor IDs for use in this context. |
| 23 (1) | V | Vendor Indicator Shall be zero if the capability is non-settable. For settable capabilities, identifies whether the capability is vendor-specific SET (1): The capability is vendor-specific CLEAR (0): The capability is defined by the TCG |
| 27:24 (4) | A | Authorization Shall be zero if the capability is non-settable. For settable capabilities, identifies the authorization required for TPM2_SetCapability(): <ul style="list-style-type: none">• 0x0: No authorization• 0x1: Platform authorization• 0x2: Owner authorization• 0x3: Platform or Owner authorization• 0x4: Endorsement authorization• 0x5: Lockout authorization• 0x6-F: Reserved |
| 30:28 (3) | Reserved | shall be zero |

(continued on next page)

(continued from previous page)

| Bit | Name | Definition |
|--------|------|--|
| 31 (1) | S | Settable SET (1): The capability is settable CLEAR (0): The capability is not settable |

6.12.1.1 Capability Authorization

The first byte of the capability value identifies the required authorization for TPM2_SetCapability(), which is enforced by the TPM. Table 27 provides a tabular view of the mapping between capability value ranges and required authorization values.

Table 27: TPM Capability Authorization

| Required authorization | Byte[0] | Capability value range |
|---------------------------------|---------|-------------------------|
| No authorization | 0x80 | 0x80000000 - 0x80FFFFFF |
| Platform authorization | 0x81 | 0x81000000 - 0x81FFFFFF |
| Owner authorization | 0x82 | 0x82000000 - 0x82FFFFFF |
| Platform or Owner authorization | 0x83 | 0x83000000 - 0x83FFFFFF |
| Endorsement authorization | 0x84 | 0x84000000 - 0x84FFFFFF |
| Lockout authorization | 0x85 | 0x85000000 - 0x85FFFFFF |
| Reserved | 0x86-8F | 0x86000000 - 0x8FFFFFFF |

6.13 TPM_PT

The Table 28 property tag 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 28: Definition of (UINT32) TPM_PT Constants

| Capability Name | Value | Comments |
|-----------------|------------|--|
| TPM_PT_NONE | 0x00000000 | indicates no property type |
| PT_GROUP | 0x00000100 | the number of properties in each group Note: NOTE The first group with any properties is group 1 (PT_GROUP * 1). Group 0 is reserved. |

(continued on next page)

(continued from previous page)

| Capability Name | Value | Comments |
|-------------------------|--------------|--|
| 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) Note: For this specification, the Family is "2.0". |
| TPM_PT_LEVEL | PT_FIXED + 1 | the level of the specification Note: For this specification, the level is zero. |
| TPM_PT_REVISION | PT_FIXED + 2 | the specification version Note: The Version value is on the title page of the specification. |
| TPM_PT_ERRATA | PT_FIXED + 3 | the errata version implemented by the TPM Note: The errata version value is on the title page of the errata document (see Clause 6.1). Note: Prior to version 185, this property was called TPM_PT_DAY_OF_YEAR |

(continued on next page)

(continued from previous page)

| Capability Name | Value | Comments |
|---------------------------|---------------|--|
| TPM_PT_YEAR | PT_FIXED + 4 | shall be zero Note: Prior to version 185, this property reported the year in which the TPM Library Specification indicated by TPM_SPEC_VERSION (or an Errata) was published. It is set to 0 to indicate that the errata level is reported in TPM_SPEC_ERRATA. |
| 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 Clause 10.11.2 and Clause 10.11.12. |
| 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 Clause 10.11.2 and Clause 10.11.12. |
| TPM_PT_INPUT_BUFFER | PT_FIXED + 13 | the maximum size of a parameter (typically, TPM2B_MAX_BUFFER) |
| TPM_PT_HR_TRANSIENT_MIN | PT_FIXED + 14 | the minimum number of transient objects that can be held in TPM RAM This minimum shall be no less than the minimum value required by the platform-specific specification to which the TPM is built. |

(continued on next page)

(continued from previous page)

| Capability Name | Value | Comments |
|----------------------------|---------------|---|
| TPM_PT_HR_PERSISTENT_MIN | PT_FIXED + 15 | the minimum number of persistent objects that can be held in TPM NV memory 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 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. 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 <i>contextID</i> values of two saved session contexts This value shall be $2^n - 1$, where n is at least 16. |
| | PT_FIXED + 21 | reserved |

(continued on next page)

(continued from previous page)

| Capability Name | Value | Comments |
|-------------------------|---------------|--|
| TPM_PT_NV_COUNTERS_MAX | PT_FIXED + 22 | <p>the maximum number of NV Indexes that are allowed to have the TPM_NT_COUNTER attribute</p> <p>Note:</p> <p>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.</p> <p>Note:</p> <p>The value zero indicates that there is no fixed maximum. The number of counter indexes is determined by the available NV memory pool.</p> |
| 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 |

(continued on next page)

(continued from previous page)

| Capability Name | Value | Comments |
|----------------------------|---------------|--|
| TPM_PT_ORDERLY_COUNT | PT_FIXED + 29 | <p>the modulus - 1 of the count for NV update of an orderly counter</p> <p>The returned value is MAX_ORDERLY_COUNT.</p> <p>This will have a value of $2^N - 1$ where $1 \leq N \leq 32$</p> <p>Note:</p> <p>An “orderly counter” is an NV Index with an TPM_NT of TPM_NV_COUNTER and TPMA_NV_ORDERLY SET.</p> <p>Note:</p> <p>When the low-order bits of a counter equal this value, an NV write occurs on the next increment.</p> |
| TPM_PT_MAX_COMMAND_SIZE | PT_FIXED + 30 | the maximum value for <i>commandSize</i> in a command |
| TPM_PT_MAX_RESPONSE_SIZE | PT_FIXED + 31 | the maximum value for <i>responseSize</i> 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 | <p>platform-specific family (a TPM_PS value)</p> <p>Note:</p> <p>The platform-specific values for the TPM_PT_PS parameters are in the relevant platform-specific specification. In the Reference Code, all of these values are 0.</p> |
| 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 |

(continued on next page)

(continued from previous page)

| Capability Name | Value | Comments |
|--------------------------|---------------|--|
| 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 (0x0000013F). |
| TPM_PT_PS_YEAR | PT_FIXED + 39 | the platform-specific TPM specification year using the CE Example: The year 2010 has a value of 0x000007DA. |
| 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(). |
| TPM_PT_FIRMWARE_SVN | PT_FIXED + 47 | the TPM vendor-specific value indicating the SVN of the firmware. This value shall be less than or equal to UINT16_MAX. |
| TPM_PT_FIRMWARE_MAX_SVN | PT_FIXED + 48 | the TPM vendor-specific value indicating the maximum value that TPM_PT_FIRMWARE_SVN may take in the future. |
| TPM_PT_ML_PARAMETER_SETS | PT_FIXED + 49 | a TPMA_ML_PARAMETER_SET indicating the supported parameters sets for ML-KEM and ML-DSA |
| | | intentionally left empty |

(continued on next page)

(continued from previous page)

| Capability Name | Value | Comments |
|------------------------|--------------|---|
| 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, which 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. Example: A valid implementation is permitted to 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, which 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. Example: A valid implementation is permitted to return 1 even if more than one authorization session could be created. |

(continued on next page)

(continued from previous page)

| Capability Name | Value | Comments |
|----------------------------|-------------|--|
| TPM_PT_HR_TRANSIENT_AVAIL | PT_VAR + 7 | <p>estimate of the number of additional transient objects that could be loaded into TPM RAM</p> <p>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.</p> <p>Example:</p> <p>A valid implementation is permitted to return 1 even if more than one transient object would fit into RAM.</p> |
| TPM_PT_HR_PERSISTENT | PT_VAR + 8 | <p>the number of persistent objects currently loaded into TPM NV memory</p> |
| TPM_PT_HR_PERSISTENT_AVAIL | PT_VAR + 9 | <p>the number of additional persistent objects that could be loaded into NV memory</p> <p>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.</p> <p>Example:</p> <p>A valid implementation is permitted to return 1 even if more than one persistent object would fit into NV memory.</p> |
| TPM_PT_NV_COUNTERS | PT_VAR + 10 | <p>the number of defined NV Indexes that have the TPM_NT_COUNTER attribute</p> |

(continued on next page)

(continued from previous page)

| Capability Name | Value | Comments |
|--------------------------|-------------|--|
| TPM_PT_NV_COUNTERS_AVAIL | PT_VAR + 11 | <p>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</p> <p>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.</p> <p>Example:</p> <p>A valid implementation is permitted to return 1 even if more than one NV counter could be defined.</p> |
| 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 (<i>failedTries</i>) |
| 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 | <p>number of milliseconds before the TPM will accept another command that will modify NV</p> <p>This value is an approximation and may go up or down over time.</p> |
| 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 |

6.14 TPM_PT_PCR

The Table 29 PCR property tag 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 29: Definition of (UINT32) TPM_PT_PCR Constants

| 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. |

(continued on next page)

(continued from previous page)

| Capability Name | Value | Comments |
|-------------------------|-----------------------------|---|
| 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 <i>pcrUpdateCounter</i> |
| 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 - 0x000000210 | reserved for the 2nd through 255th TPM_PT_PCR_POLICY and TPM_PT_PCR_AUTH values |

(continued on next page)

(continued from previous page)

| Capability Name | Value | Comments |
|-----------------|------------|---|
| 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 <i>moreData</i> parameter to NO. Note: This is an implementation-specific value. The value shown reflects the Reference Code implementation. |

6.15 TPM_PS

The platform specific values in Table 30 are used for the TPM_PT_PS_FAMILY_INDICATOR.

Table 30 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.

Table 30: Definition of (UINT32) TPM_PS Constants

| 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 |

(continued on next page)

(continued from previous page)

| Capability Name | Value | Comments |
|-----------------------|------------|---|
| 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) |

6.16 TPM_PUB_KEY

The Table 31 public key constants are used in `TPM2_GetCapability` (*capability* == `TPM_CAP_PUB_KEYS`) to indicate the public key to be returned.

Note:

This is useful for public keys that are stored in the TPM but not used as TPM objects (and therefore not readable with `TPM2_ReadPublic()`). Currently, this is only used to read and attest TPM secure channel public keys, which may be used e.g. by a TPM's secure transport layer for secure channel establishment. This also allows computation of the Name of the TPM secure channel public key, which is used in `TPM2_PolicyTransportSPDM()`. The public key is returned instead of the Name to allow the object attributes to be read.

Note:

Typically, only one TPM secure channel public key will be available. In this case, only the `TPM_PUB_KEY_TPM_SPDM_00` property would be defined. `TPM_RC_VALUE` is returned for not supported property values.

Table 31: Definition of (UINT32) `TPM_PUB_KEY` Constants

| Property Name | Value | Comments |
|--------------------------------------|------------|---|
| <code>TPM_PUB_KEY_TPM_SPDM_00</code> | 0x00000000 | Start of the property range for TPM SPDM authentication public keys |
| <code>TPM_PUB_KEY_TPM_SPDM_FF</code> | 0x000000FF | End of the property range for TPM SPDM authentication public keys |

7 Handles

7.1 Introduction

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

Table 32: Definition of Types for TPM_HANDLE

| 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.

7.2 TPM_HT

The 32-bit handle type space is divided into 256 regions of equal size with 2^{24} 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 33.

Not all values of TPM_HT have meaning in all contexts of TPM handles. Some of the values of TPM_HT only have meaning in that context. Values that have meaning in the general (non-TPM2_GetCapability) context of a handle are marked with an X in the “Gen” column in Table 33. Values that have meaning in the TPM2_GetCapability context are marked with an X in the “Cap” column in Table 33.

Table 33: Definition of (UINT8) TPM_HT Constants

| Name | Value | Gen | Cap | Comments |
|-----------------------|-------|-----|-----|---|
| TPM_HT_PCR | 0x00 | X | X | PCR - consecutive numbers, starting at 0, that reference the PCRs A platform-specific specification will set the minimum number of PCR and an implementation may have more. |
| TPM_HT_NV_INDEX | 0x01 | X | X | NV Index - assigned by the caller |
| TPM_HT_HMAC_SESSION | 0x02 | X | | HMAC Authorization Session - assigned by the TPM when the session is created |
| TPM_HT_LOADED_SESSION | 0x02 | | X | Loaded Authorization Session - loaded sessions of type TPM_HT_HMAC_SESSION or TPM_HT_POLICY_SESSION |
| TPM_HT_POLICY_SESSION | 0x03 | X | | Policy Authorization Session - assigned by the TPM when the session is created |

(continued on next page)

(continued from previous page)

| Name | Value | Gen | Cap | Comments |
|----------------------|-------|-----|-----|---|
| TPM_HT_SAVED_SESSION | 0x03 | | X | Saved Authorization Session - saved sessions of type type TPM_HT_HMAC_SESSION or TPM_HT_POLICY_SESSION This type references saved authorization session contexts for which the TPM is maintaining tracking information. |
| TPM_HT_EXTERNAL_NV | 0x11 | X | | External NV Index - assigned by the caller NOTE: Added in version 1.83. |
| TPM_HT_PERMANENT_NV | 0x12 | X | | Permanent NV Index - assigned by a platform specific specification NOTE: Added in version 1.83. |
| TPM_HT_PERMANENT | 0x40 | X | X | Permanent Values - assigned by this specification in Table 34 |
| TPM_HT_TRANSIENT | 0x80 | X | X | 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 | X | X | Persistent Objects - assigned by the TPM when a loaded transient object is made persistent |
| TPM_HT_AC | 0x90 | X | | 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:

If a TPM is only able to hold 4 transient objects in internal memory, it could 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:

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.

7.3 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.

7.4 TPM_RH

Table 34 lists the architecturally defined permanent handles that cannot be changed. The handles include authorization handles, and special handles.

Table 34: Definition of (TPM_HANDLE) TPM_RH Constants

| Name | Value | Type | Comments |
|--------------------|------------|---------|--|
| TPM_RH_FIRST | 0x40000000 | R | |
| TPM_RH_SRK | 0x40000000 | R | not used |
| TPM_RH_OWNER | 0x40000001 | K, A, P | handle references the Storage Primary Seed (SPS), the <i>ownerAuth</i> , and the <i>ownerPolicy</i> |
| TPM_RH_REVOKE | 0x40000002 | R | not used |
| TPM_RH_TRANSPORT | 0x40000003 | R | not used |
| TPM_RH_OPERATOR | 0x40000004 | R | not used |
| TPM_RH_ADMIN | 0x40000005 | R | not used |
| TPM_RH_EK | 0x40000006 | R | not used |
| TPM_RH_NULL | 0x40000007 | K, A, P | a handle associated with the NULL hierarchy, an EmptyAuth <i>authValue</i> , and an Empty Policy <i>authPolicy</i> . |
| 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), <i>endorsementAuth</i> , and <i>endorsementPolicy</i> |

(continued on next page)

(continued from previous page)

| Name | Value | Type | Comments |
|-----------------------------|------------|---------|---|
| TPM_RH_PLATFORM | 0x4000000C | K, A, P | references the Platform Primary Seed (PPS), <i>platformAuth</i> , and <i>platformPolicy</i> |
| 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_FW_OWNER | 0x40000140 | K | references the firmware-limited Owner hierarchy |
| TPM_RH_FW_ENDORSEMENT | 0x40000141 | K | references the firmware-limited Endorsement hierarchy |
| TPM_RH_FW_PLATFORM | 0x40000142 | K | references the firmware-limited Platform hierarchy |
| TPM_RH_FW_NULL | 0x40000143 | K | references the firmware-limited NULL hierarchy |
| TPM_RH_SVN_OWNER_BASE | 0x40010000 | K | references the base SVN-limited Owner hierarchy. The low 2 bytes represent the minimum SVN value to which the hierarchy is limited. |
| TPM_RH_SVN_ENDORSEMENT_BASE | 0x40020000 | K | references the base SVN-limited Endorsement hierarchy. The low 2 bytes represent the minimum SVN value to which the hierarchy is limited. |
| TPM_RH_SVN_PLATFORM_BASE | 0x40030000 | K | references the base SVN-limited Platform hierarchy. The low 2 bytes represent the minimum SVN value to which the hierarchy is limited. |
| TPM_RH_SVN_NULL_BASE | 0x40040000 | K | references the base SVN-limited NULL hierarchy. The low 2 bytes represent the minimum SVN value to which the hierarchy is limited. |

(continued on next page)

(continued from previous page)

| Name | Value | Type | Comments |
|--|------------|------|--|
| TPM_RH_LAST | 0x4004FFFF | 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: 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. | | | |

7.5 TPM_HC

The handle value constants in Table 35 are used to define many of the interface data types.

These values, which indicate ranges, are informative and may be changed by an implementation. The TPM will always return the correct handle type as described in Clause 7.2 and Table 33:

- 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 Code, 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 35 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 35: Definition of (TPM_HANDLE) TPM_HC Constants

| 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_EXTERNAL_NV | (TPM_HT_EXTERNAL_NV « HR_SHIFT) | |
| HR_PERMANENT_NV | (TPM_HT_PERMANENT_NV « 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 |

(continued on next page)

(continued from previous page)

| Name | Value | Comments |
|-----------------------|--|---|
| 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 with 32-bit attributes |
| NV_INDEX_LAST | (NV_INDEX_FIRST + 0x00FFFFFF) | last allowed NV Index with 32-bit attributes |
| EXTERNAL_NV_FIRST | (HR_EXTERNAL_NV + 0) | first external NV Index |
| EXTERNAL_NV_LAST | (NV_EXTERNAL_FIRST + 0x00FFFFFF) | last external NV Index |
| PERMANENT_NV_FIRST | (HR_PERMANENT_NV + 0) | first permanent NV Index |
| PERMANENT_NV_LAST | (NV_PERMANENT_NV_FIRST + 0x00FFFFFF) | last permanent NV Index |
| PERMANENT_FIRST | TPM_RH_FIRST | |
| PERMANENT_LAST | TPM_RH_LAST | |
| SVN_OWNER_FIRST | (TPM_RH_SVN_OWNER_BASE + 0x0000) | |
| SVN_OWNER_LAST | (TPM_RH_SVN_OWNER_BASE + 0xFFFF) | |
| SVN_ENDORSEMENT_FIRST | (TPM_RH_SVN_ENDORSEMENT_BASE + 0x0000) | |
| SVN_ENDORSEMENT_LAST | (TPM_RH_SVN_ENDORSEMENT_BASE + 0xFFFF) | |
| SVN_PLATFORM_FIRST | (TPM_RH_SVN_PLATFORM_BASE + 0x0000) | |
| SVN_PLATFORM_LAST | (TPM_RH_SVN_PLATFORM_BASE + 0xFFFF) | |
| SVN_NULL_FIRST | (TPM_RH_SVN_NULL_BASE + 0x0000) | |
| SVN_NULL_LAST | (TPM_RH_SVN_NULL_BASE + 0xFFFF) | |
| HR_NV_AC | ((TPM_HT_NV_INDEX « HR_SHIFT) + 0xD00000) | AC aliased NV Index |

(continued on next page)

(continued from previous page)

| Name | Value | Comments |
|-------------|-------------------------|--|
| 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 |

8 Attribute Structures

8.1 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.

8.2 TPMA_ALGORITHM

The Table 36 structure defines the attributes of an algorithm.

Deprecated:

All bits in TPMA_ALGORITHM were deprecated in version 185. See Part 0.



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 36: 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 | |

(continued on next page)

(continued from previous page)

| Bit | Name | Definition |
|-------|------------|--|
| 8 | signing | SET (1): a signing algorithm. The setting of <i>asymmetric</i> , <i>symmetric</i> , and <i>hash</i> will indicate the type of signing algorithm. CLEAR (0): not a signing algorithm |
| 9 | encrypting | SET (1): an encryption/decryption algorithm. The setting of <i>asymmetric</i> , <i>symmetric</i> , and <i>hash</i> 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 | |

8.3 TPMA_OBJECT

8.3.1 Introduction

This object attribute structure in Table 37 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.

8.3.2 Structure Definition

Table 37: 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: <i>fixedTPM</i> does not indicate that key material resides on a single TPM (see <i>sensitiveDataOrigin</i>). |
| 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(). See Clause 14.3 for a use case. |

(continued on next page)

(continued from previous page)

| Bit | Name | Definition |
|-------|----------------------|---|
| 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 <i>authValue</i> . CLEAR (0): A portion of the sensitive data, other than the <i>authValue</i> , 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 <i>authValue</i> 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 <i>authValue</i> of the object or a policy session. |
| 8 | firmwareLimited | SET (1): The object exists only within a firmware-limited hierarchy. CLEAR (0): The object can exist outside a firmware-limited hierarchy. |
| 9 | svnLimited | SET (1): The object exists only within an SVN-limited hierarchy. CLEAR (0): The object can exist outside an SVN-limited hierarchy. |
| 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 <i>symmetricAlg</i> shall not be TPM_ALG_NULL and <i>newParentHandle</i> 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 <i>restricted</i> SET. CLEAR (0): Key usage is not restricted to use on special formats. |

(continued on next page)

(continued from previous page)

| Bit | Name | Definition |
|-------|----------------|--|
| 17 | decrypt | <p>SET (1): The private portion of the key may be used to decrypt.</p> <p>CLEAR (0): The private portion of the key may not be used to decrypt.</p> <p>Note: Version 185 deprecated support for keys with both <i>sign</i> and <i>decrypt</i> SET. See Part 0.</p> |
| 18 | sign / encrypt | <p>SET (1): For a symmetric block cipher key, the private portion of the key may be used to encrypt. For other keys, the private portion of the key may be used to sign.</p> <p>CLEAR (0): The private portion of the key may not be used to sign or encrypt.</p> <p>Note: Version 185 deprecated support for keys with both <i>sign</i> and <i>decrypt</i> SET. See Part 0.</p> |
| 19 | x509sign | <p>SET (1): An asymmetric key that may not be used to sign with TPM2_Sign(), TPM2_SignDigest(), or TPM2_SignSequenceComplete().</p> <p>CLEAR (0): A key that may be used with TPM2_Sign(), TPM2_SignDigest, or TPM2_SignSequenceComplete() if <i>sign</i> is SET.</p> <p>Note: This attribute only has significance if <i>sign</i> is SET.</p> <p>Note: This attribute does not prevent the key from being used with other commands, such as TPM2_Quote().</p> |
| 31:20 | Reserved | shall be zero |

8.3.3 Attribute Descriptions

8.3.3.1 Introduction

The following remaining paragraphs in this clause 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()

(continued on next page)

(continued from previous page)

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 37 and the reserved attributes check.

For any consistency error of attributes in `TPMA_OBJECT`, the TPM shall return `TPM_RC_ATTRIBUTES`.

8.3.3.2 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.

Creation If *fixedTPM* is SET in the object's parent, then *fixedTPM* shall equal *fixedParent* 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

8.3.3.3 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

8.3.3.4 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 |

8.3.3.5 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:

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 <i>inSensitive.sensitive.data.size</i> is zero, then this attribute shall be SET in the template; otherwise, it shall be CLEAR in the template. |
|-----------------|--|

Note:

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

Note:

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 |

8.3.3.6 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:

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:

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 |

8.3.3.7 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 the HMAC or password authorizations may not be used for ADMIN role authorizations.

Note:

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:

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* can 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 <i>template</i> |
| Load | may be SET or CLEAR |
| Import | may be SET or CLEAR |
| External | may be SET or CLEAR |

8.3.3.8 Bit[8] - *firmwareLimited*

If SET, the object exists only within a firmware-limited hierarchy.

If CLEAR, the object can exist outside a firmware-limited hierarchy.

Creation May only be SET if all of the following are true:

- *fixedTPM* is SET in *template*
- *firmwareLimited* is SET in the object's parent

Note:

For a Primary Object in a Firmware-limited hierarchy, the parent is considered to have *firmwareLimited* SET.

Load may be SET or CLEAR

Import shall be CLEAR

External shall be CLEAR

8.3.3.9 Bit[9] - *svnLimited*

If SET, the object exists only within an SVN-limited hierarchy.

If CLEAR, the object can exist outside a SVN-limited hierarchy.

Creation May only be SET if all of the following are true:

- *fixedTPM* is SET in *template*
- *svnLimited* is SET in the object's parent

Note:

For a Primary Object in an SVN-limited hierarchy, the parent is considered to have *svnLimited* SET.

Load may be SET or CLEAR

Import shall be CLEAR

External shall be CLEAR

8.3.3.10 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

8.3.3.11 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:

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:

This requirement means that *encryptedDuplication* cannot 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:

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:

This requirement forces all duplicable objects in a duplication group to have the same *encryptedDuplication* setting.

| | |
|-----------------|---|
| Creation | shall be CLEAR if <i>fixedTPM</i> is SET. If <i>fixedTPM</i> is CLEAR, then this attribute shall have the same value as its parent unless <i>fixedTPM</i> is SET in the object's parent, in which case, it may be SET or CLEAR. |
| Load | shall be CLEAR if <i>fixedTPM</i> is SET. If <i>fixedTPM</i> is CLEAR, then this attribute shall have the same value as its parent, unless <i>fixedTPM</i> is SET the parent, in which case, it may be SET or CLEAR. |
| Import | if <i>fixedTPM</i> 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. |

8.3.3.12 Bit[16] - *restricted*

This attribute modifies the *decrypt* and *sign* attributes of an object.

Note:

A key with this object CLEAR cannot be a parent for another object.

| | |
|-----------------|---|
| Creation | shall be CLEAR in <i>template</i> if neither sign nor decrypt is SET in <i>template</i> . |
| Load | shall be CLEAR if neither sign nor decrypt is SET in the object |
| Import | may be SET or CLEAR |
| External | shall be CLEAR |

8.3.3.13 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:

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:

A key with this attribute SET can 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 <i>template</i> |
| Load | may be SET or CLEAR |
| Import | may be SET or CLEAR |
| External | may be SET or CLEAR |

8.3.3.14 Bit[18] - *sign / encrypt*

When SET, the private portion of this key may be used to sign a digest if the key is an asymmetric key or a symmetric MAC key, or to encrypt a block of data if the key is a symmetric block cipher 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 block cipher 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:

Some asymmetric algorithms are permitted to 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:

A key with this attribute SET cannot be a parent for another object.

| | |
|-----------------|---|
| Creation | shall not be SET if <i>decrypt</i> and <i>restricted</i> are both SET (continued on next page) |
|-----------------|---|

(continued from previous page)

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

8.3.3.15 Bit[19] - *x509sign*

When SET, the private portion of the asymmetric key may not be used as the signing key in `TPM2_Sign()`, `TPM2_SignSequenceComplete()`, and `TPM2_SignDigest()`. This restriction is to ensure that the only digest signed by this key is a digest of a structure that is specific to the TPM or an x509 certificate.

Note:

This attribute does not limit the use of the key in any command other than `TPM2_Sign()`, `TPM2_SignSequenceComplete()`, and `TPM2_SignDigest()`.

This attribute was added in version 1.59.

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

8.4 TPMA_SESSION

This session attribute octet (see Table 38) 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 version, if *audit* is CLEAR, *auditExclusive* must be CLEAR in the command and will be CLEAR in the response. In a future version, this bit may have a different meaning if *audit* is CLEAR. See “Exclusive Audit Session” clause in TPM 2.0 Part 1.

In this version, if *audit* is CLEAR, *auditReset* must be clear in the command and will be CLEAR in the response. In a future version, 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 38: Definition of (UINT8) TPMA_SESSION Bits

| Bit | Name | Meaning |
|-----|------|---------|
|-----|------|---------|

(continued on next page)

(continued from previous page)

| Bit | Name | Meaning |
|-----|-----------------|--|
| 0 | continueSession | <p>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.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>This attribute has no effect if the command does not complete successfully.</p> |
| 1 | auditExclusive | <p>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 <i>audit</i> attribute is SET (TPM_RC_ATTRIBUTES).</p> <p>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.</p> |
| 2 | auditReset | <p>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 <i>audit</i> attribute is SET (TPM_RC_ATTRIBUTES).</p> <p>CLEAR (0): In a command, indicates that the audit digest should not be initialized.</p> <p>This bit is always CLEAR in a response.</p> |
| 4:3 | Reserved | shall be CLEAR |

(continued on next page)

(continued from previous page)

| Bit | Name | Meaning |
|-----|---------|---|
| 5 | decrypt | <p>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.</p> <p>CLEAR (0): Session not used for encryption.</p> <p>For a password authorization, this attribute will be CLEAR in both the command and response.</p> <p>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.</p> <p>This attribute may be SET in combination with any other session attributes.</p> |
| 6 | encrypt | <p>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.</p> <p>CLEAR (0): Session not used for encryption.</p> <p>For a password authorization, this attribute will be CLEAR in both the command and response.</p> <p>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.</p> |
| 7 | audit | <p>SET (1): In a command or response, this setting indicates that the session is for audit and that <i>auditExclusive</i> and <i>auditReset</i> have meaning. This session may also be used for authorization, encryption, or decryption. The <i>encrypted</i> and <i>encrypt</i> fields may be SET or CLEAR.</p> <p>CLEAR (0): Session is not used for audit.</p> <p>If SET in the command, then this attribute will be SET in the response.</p> |

8.5 TPMA_LOCALITY

In a TPMS_CREATION_DATA Structure, the Table 39 locality attribute 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:

TPM_LOC_TWO would indicate that only locality 2 is authorized.

Example:

TPM_LOC_ONE + TPM_LOC_TWO would indicate that locality 1 or 2 is authorized.

Example:

TPM_LOC_FOUR + TPM_LOC_THREE would indicate that localities 3 or 4 are authorized.

Example:

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 39: Definition of (UINT8) TPMA_LOCALITY Bits

| 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 |

8.6 TPMA_PERMANENT

The attributes in the Table 40 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 40: Definition of (UINT32) TPMA_PERMANENT Bits

| Bit | Parameter | Description |
|-----|--------------------|--|
| 0 | ownerAuthSet | SET (1): TPM2_HierarchyChangeAuth() with <i>ownerAuth</i> has been executed since the last TPM2_Clear(). CLEAR (0): <i>ownerAuth</i> has not been changed since TPM2_Clear(). |
| 1 | endorsementAuthSet | SET (1): TPM2_HierarchyChangeAuth() with <i>endorsementAuth</i> has been executed since the last TPM2_Clear(). CLEAR (0): <i>endorsementAuth</i> has not been changed since TPM2_Clear(). |

(continued on next page)

(continued from previous page)

| Bit | Parameter | Description |
|-------|-----------------|--|
| 2 | lockoutAuthSet | SET (1): TPM2_HierarchyChangeAuth() with <i>lockoutAuth</i> has been executed since the last TPM2_Clear(). CLEAR (0): <i>lockoutAuth</i> 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 <i>failedTries</i> is equal to <i>maxTries</i> . |
| 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 | |

8.7 TPMA_STARTUP_CLEAR

The Table 41 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.

readOnly is CLEAR on TPM Reset or TPM Restart and preserved by TPM Resume. If *phEnable*, *shEnable*, or *ehEnable* is CLEAR, then Read-Only mode is not applicable to the corresponding hierarchy.

Some attributes may be changed as the result of specific Protected Capabilities.

Table 41: Definition of (UINT32) TPMA_STARTUP_CLEAR Bits

| Bit | Parameter | Description |
|-----|-----------|-------------|
|-----|-----------|-------------|

(continued on next page)

(continued from previous page)

| Bit | Parameter | Description |
|-----|-----------|--|
| 0 | phEnable | <p>SET (1): The platform hierarchy is enabled and <i>platformAuth</i> or <i>platformPolicy</i> may be used for authorization.</p> <p>CLEAR (0): <i>platformAuth</i> and <i>platformPolicy</i> may not be used for authorizations, and objects in the platform hierarchy, including persistent objects, cannot be used.</p> <p>Note: See “TPM2_HierarchyControl()” in TPM 2.0 Part 3 for details on changing this attribute.</p> |
| 1 | shEnable | <p>SET (1): The Storage hierarchy is enabled and <i>ownerAuth</i> or <i>ownerPolicy</i> may be used for authorization. NV indices defined using owner authorization are accessible.</p> <p>CLEAR (0): <i>ownerAuth</i> and <i>ownerPolicy</i> may not be used for authorizations, and objects in the Storage hierarchy, persistent objects, and NV indices defined using owner authorization cannot be used.</p> <p>Note: See “TPM2_HierarchyControl()” in TPM 2.0 Part 3 for details on changing this attribute.</p> |
| 2 | ehEnable | <p>SET (1): The EPS hierarchy is enabled and Endorsement Authorization may be used to authorize commands.</p> <p>CLEAR (0): Endorsement Authorization may not be used for authorizations, and objects in the endorsement hierarchy, including persistent objects, cannot be used.</p> <p>Note: See “TPM2_HierarchyControl()” in TPM 2.0 Part 3 for details on changing this attribute.</p> |

(continued on next page)

(continued from previous page)

| Bit | Parameter | Description |
|------|------------|--|
| 3 | phEnableNV | <p>SET (1): NV indices that have TPMA_NV_PLATFORMCREATE SET may be accessed (e.g., read, written, locked, used in a policy). The platform can define and undefine indices if <i>phEnable</i> is SET.</p> <p>Note: Access is independent of the state of <i>phEnable</i>.</p> <p>CLEAR (0): NV indices that have TPMA_NV_PLATFORMCREATE SET may not be accessed (TPM_RC_HANDLE). The platform cannot define (TPM_RC_HIERARCHY) or undefine (TPM_RC_HANDLE) indices.</p> <p>Note: See “TPM2_HierarchyControl()” in TPM 2.0 Part 3 for details on changing this attribute.</p> <p>Note: “read” refers to TPM2_NV_Read(), TPM2_NV_ReadPublic(), TPM_NV_Certify(), and TPM2_PolicyNV() “write” refers TPM2_NV_Write(), TPM2_NV_Increment(), TPM2_NV_Extend(), and TPM2_NV_SetBits()</p> <p>Note: The TPM must query the index TPMA_NV_PLATFORMCREATE attribute to determine whether <i>phEnableNV</i> 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.</p> |
| 4 | readOnly | <p>SET (1): All enabled hierarchies, including the NULL hierarchy, are Read-Only.</p> <p>CLEAR (0): All enabled hierarchies can be modified.</p> <p>Note: See TPM2_ReadOnlyControl() in TPM 2.0 Part 3 for details on changing this attribute.</p> |
| 30:5 | Reserved | shall be zero |

(continued on next page)

(continued from previous page)

| Bit | Parameter | Description |
|-----|-----------|---|
| 31 | orderly | <p>SET (1): The TPM received a TPM2_Shutdown() and a matching TPM2_Startup().</p> <p>CLEAR (0): TPM2_Startup(TPM_SU_CLEAR) was not preceded by a TPM2_Shutdown() of any type.</p> <p>Note:</p> <p>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).</p> |

8.8 TPMA_MEMORY

The Table 42 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 42: Definition of (UINT32) TPMA_MEMORY Bits

| Bit | Name | Definition |
|------|-------------------|--|
| 0 | sharedRAM | <p>SET (1): indicates that the RAM memory used for authorization session contexts is shared with the memory used for transient objects</p> <p>CLEAR (0): indicates that the memory used for authorization sessions is not shared with memory used for transient objects</p> |
| 1 | sharedNV | <p>SET (1): indicates that the NV memory used for persistent objects is shared with the NV memory used for NV Index values</p> <p>CLEAR (0): indicates that the persistent objects and NV Index values are allocated from separate sections of NV</p> |
| 2 | objectCopiedToRam | <p>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.</p> <p>CLEAR (0): indicates that the TPM does not use transient-object slots when persistent objects are referenced</p> |
| 31:3 | Reserved | shall be zero |

8.9 TPMA_CC

8.9.1 Introduction

The Table 43 command code attribute 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).

Note:

Commands to the TPM use the related TPM_CC type, which has the same bit assignments with all bits reserved/zero except the *commandIndex* field and *V* attribute.

8.9.2 Structure Definition

Table 43: Definition of (UINT32) TPMA_CC Bits

| 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 <i>flushed</i> 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 |

8.9.3 Field Descriptions

8.9.3.1 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.

8.9.3.2 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.

8.9.3.3 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 can be combined with other decorations such as “{NV}” in which case the decoration would be “{NV E}.”

8.9.3.4 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 can 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 behavior 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.”

8.9.3.5 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).

8.9.3.6 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.

8.9.3.7 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.

8.9.3.8 Bits[31:30] - Res

This field is reserved for system software. This field is required to be zero for a command to the TPM.

8.10 TPMA_MODES

The Table 44 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.

FIPS_140_3_INDICATOR values are

(00): service belongs to category 2, 3 or 5 (non-security relevant services)

(01): service belongs to category 1 (approved security services)

(10): service belongs to category 4 (non-approved security services)

(11): reserved value

Note:

The categories 1 to 5 are defined in FIPS140-3 IG 2.4.C

Note:

The FIPS_140_3_INDICATOR was added in version 1.83.

Table 44: Definition of (UINT32) TPMA_MODES Bits

| 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. |

(continued on next page)

(continued from previous page)

| Bit | Name | Definition |
|------|----------------------|--|
| 1 | FIPS_140_3 | SET (1): indicates that the TPM is designed to comply with all of the FIPS 140-3 requirements at Level 1 or higher. Note: In the Reference Code, this bit is CLEAR. |
| 3:2 | FIPS_140_3_INDICATOR | indicates the FIPS 140-3 category of the service provided by the last command successfully executed before TPM2_GetCapability (<i>capability</i> == TPM_CAP_TPM_PROPERTIES, <i>property</i> == TPM_PT_MODES) command This attribute is only meaningful if FIPS_140_3 is SET. Note: In the Reference Code, these bits are set to (00) and have no meaning. |
| 31:4 | Reserved | shall be zero |

8.11 TPMA_X509_KEY_USAGE

Deprecated:

TPM2_CertifyX509() was deprecated in version 184. See Part 0.



The Table 45 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 (Subject Key, *objectHandle*) 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 complementary 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.

Note:

keyEncipherment indicates that the key is restricted to encrypting another key. It maps to a TPM parent key. *dataEncipherment* indicates that the key can be used for any encryption.

Table 45: Definition of (UINT32) TPMA_X509_KEY_USAGE Bits

| Bit | Attribute | Requirements |
|------|-----------------------------------|---|
| 22:0 | Reserved | |
| 23 | decipherOnly | Attributes. <i>decrypt</i> SET |
| 24 | encipherOnly | Attributes. <i>decrypt</i> SET |
| 25 | CRLSign | Attributes. <i>sign</i> SET |
| 26 | keyCertSign | Attributes. <i>sign</i> SET |
| 27 | keyAgreement | Attributes. <i>decrypt</i> SET |
| 28 | dataEncipherment | Attributes. <i>decrypt</i> SET |
| 29 | keyEncipherment | asymmetric key with Attributes. <i>decrypt</i> and Attributes. <i>restricted</i> SET - key has the attributes of a parent key |
| 30 | non-repudiation/contentCommitment | Attributes. <i>fixedTPM</i> SET in Subject Key (<i>objectHandle</i>) |
| 31 | digitalSignature | Attributes. <i>sign</i> SET in Subject Key (<i>objectHandle</i>) |

8.12 TPMA_ACT

This Table 46 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). Once SET, *signaled* will remain SET until CLEAR’ed by `TPM2_ACT_SetTimeout()`, TPM Reset, or TPM Restart. *signaled* is preserved across TPM Resume.

The *preservedSignaled* value indicates whether a system reset may have been initiated due to an ACT timeout. As there is no defined relationship between a specific ACT and a system reset, *preservedSignaled* behavior is the same for all ACT.

On a TPM Reset or Restart, *preservedSignaled* is CLEAR for all ACT. On a TPM Resume, the state of each ACT *signaled* is copied to the corresponding *preservedSignaled*.

Note:

An ACT *signaled* value is CLEAR when the ACT is next accessed by `TPM2_ACT_SetTimeout()` with a non-zero *startTimeout*.

Table 46: 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 | preserves the state of signaled, depending on the power cycle On a TPM Reset or Restart, <i>preservedSignaled</i> is CLEAR for all ACT. On a TPM Resume, the state of each ACT signaled is copied to the corresponding <i>preservedSignaled</i> . |

(continued on next page)

(continued from previous page)

| Bit | Name | Definition |
|------|----------|---------------|
| 31:2 | Reserved | shall be zero |

8.13 TPMA_ML_PARAMETER_SET

This Table 47 attribute is used to report the supported ML-KEM and ML-DSA parameter sets, as well as support for *allowExternalMu*. This structure may be read using `TPM2_GetCapability(capability == TPM_CAP_TPM_PROPERTIES, property == TPM_PT_ML_PARAMETER_SETS)`.

Table 47: Definition of (UINT32) TPMA_ML_PARAMETER_SET Bits

| Bit | Name | Definition |
|------|------------|---|
| 0 | mlKem_512 | Indicates support for TPM_MLKEM_512 |
| 1 | mlKem_768 | Indicates support for TPM_MLKEM_768 |
| 2 | mlKem_1024 | Indicates support for TPM_MLKEM_1024 |
| 3 | mlDsa_44 | Indicates support for TPM_MLDSA_44 |
| 4 | mlDsa_65 | Indicates support for TPM_MLDSA_65 |
| 5 | mlDsa_87 | Indicates support for TPM_MLDSA_87 |
| 6 | extMu | Indicates support for <i>allowExternalMu</i> for ML-DSA |
| 31:7 | Reserved | |

9 Interface Types

9.1 Introduction

This clause 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.

9.2 TPMI_YES_NO

The Table 48 interface type is used in place of a Boolean type in order to eliminate ambiguity in the handling of an 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 48: Definition of (BYTE) TPMI_YES_NO Type

| Value | Description |
|---------------|--------------|
| NO | a value of 0 |
| YES | a value of 1 |
| #TPM_RC_VALUE | |

9.3 TPMI_DH_OBJECT

The Table 49 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 49: 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 | |

9.4 TPMI_DH_PARENT

The Table 50 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 50: Definition of (TPM_HANDLE) TPML_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 | NULL hierarchy |
| TPM_RH_FW_OWNER | Firmware-limited Storage hierarchy |
| TPM_RH_FW_PLATFORM | Firmware-limited Platform hierarchy |
| TPM_RH_FW_ENDORSEMENT | Firmware-limited Endorsement hierarchy |
| TPM_RH_FW_NULL | Firmware-limited NULL hierarchy |
| {SVN_OWNER_FIRST:SVN_OWNER_LAST} | SVN-limited Storage hierarchies |
| {SVN_PLATFORM_FIRST:SVN_PLATFORM_LAST} | SVN-limited Platform hierarchies |
| {SVN_ENDORSEMENT_FIRST : SVN_ENDORSEMENT_LAST} | SVN-limited Endorsement hierarchies |
| {SVN_NULL_FIRST:SVN_NULL_LAST} | SVN-limited NULL hierarchies |
| #TPM_RC_VALUE | |

9.5 TPML_DH_PERSISTENT

The Table 51 TPML_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 51: Definition of (TPM_HANDLE) TPML_DH_PERSISTENT Type

| Values | Comments |
|------------------------------------|--------------------------------------|
| {PERSISTENT_FIRST:PERSISTENT_LAST} | allowed range for persistent objects |
| #TPM_RC_VALUE | |

9.6 TPML_DH_ENTITY

The Table 52 TPML_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 52: Definition of (TPM_HANDLE) TPML_DH_ENTITY Type

| Values | Comments |
|--------------------|----------|
| TPM_RH_OWNER | |
| TPM_RH_ENDORSEMENT | |

(continued on next page)

(continued from previous page)

| Values | Comments |
|--------------------------------------|---|
| 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 | |

9.7 TPMI_DH_PCR

The Table 53 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:

Typically, the 0th PCR will have a handle value of zero.

Table 53: Definition of (TPM_HANDLE) TPMI_DH_PCR Type

| Values | Comments |
|----------------------|-------------------|
| {PCR_FIRST:PCR_LAST} | |
| +TPM_RH_NULL | conditional value |
| #TPM_RC_VALUE | |

9.8 TPMI_SH_AUTH_SESSION

The Table 54 TPMI_SH_AUTH_SESSION interface type is TPM-defined values that are used to indicate that the handle refers to an authorization session.

Table 54: Definition of (TPM_HANDLE) TPMI_SH_AUTH_SESSION Type

| 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 |

9.9 TPMI_SH_HMAC

The Table 55 interface type is used for an authorization handle when the authorization session uses an HMAC.

Table 55: Definition of (TPM_HANDLE) TPMI_SH_HMAC Type

| 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 |

9.10 TPMI_SH_POLICY

The Table 56 interface type is used for a policy handle when it appears in a policy command.

Table 56: Definition of (TPM_HANDLE) TPMI_SH_POLICY Type

| 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 |

9.11 TPMI_DH_CONTEXT

The Table 57 type defines the handle values that may be used in TPM2_ContextSave() or TPM2_Flush().

Table 57: 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 | |

9.12 TPMI_DH_SAVED

The Table 58 type defines the handle values that may be used in TPM2_ContextSave() or TPM2_FlushContext().

Table 58: 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 |

(continued on next page)

(continued from previous page)

| Values | Comments |
|---------------|--|
| 0x80000001 | a sequence object |
| 0x80000002 | a transient object with the <i>stClear</i> attribute SET |
| #TPM_RC_VALUE | |

9.13 TPMI_RH_HIERARCHY

The Table 59 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 59: 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 | NULL hierarchy |
| TPM_RH_FW_OWNER | Firmware-limited Storage hierarchy |
| TPM_RH_FW_PLATFORM | Firmware-limited Platform hierarchy |
| TPM_RH_FW_ENDORSEMENT | Firmware-limited Endorsement hierarchy |
| TPM_RH_FW_NULL | Firmware-limited NULL hierarchy |
| {SVN_OWNER_FIRST:SVN_OWNER_LAST} | SVN-limited Storage hierarchies |
| {SVN_PLATFORM_FIRST:SVN_PLATFORM_LAST} | SVN-limited Platform hierarchies |
| {SVN_ENDORSEMENT_FIRST:SVN_ENDORSEMENT_LAST} | SVN-limited Endorsement hierarchies |
| {SVN_NULL_FIRST:SVN_NULL_LAST} | SVN-limited NULL hierarchies |
| #TPM_RC_VALUE | response code returned when the unmarshaling of this type fails |

9.14 TPMI_RH_ENABLES

The Table 60 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 60: 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 |

(continued on next page)

(continued from previous page)

| Values | Comments |
|---------------|---|
| +TPM_RH_NULL | NULL hierarchy |
| #TPM_RC_VALUE | response code returned when the unmarshaling of this type fails |

9.15 TPMI_RH_HIERARCHY_AUTH

The Table 61 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 61: Definition of (TPM_HANDLE) TPMI_RH_HIERARCHY_AUTH Type

| Values | Comments |
|--------------------|---|
| TPM_RH_OWNER | Storage hierarchy |
| TPM_RH_PLATFORM | Platform hierarchy |
| TPM_RH_ENDORSEMENT | Endorsement hierarchy |
| TPM_RH_LOCKOUT | Lockout Authorization |
| +TPM_RH_NULL | NULL hierarchy |
| #TPM_RC_VALUE | response code returned when the unmarshaling of this type fails |

9.16 TPMI_RH_HIERARCHY_POLICY

The Table 62 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 62: Definition of (TPM_HANDLE) TPMI_RH_HIERARCHY_POLICY Type

| 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 |

9.17 TPMI_RH_BASE_HIERARCHY

The Table 63 interface type is used as the type of a handle in a command when the handle is required to be one of the base hierarchy selectors. This type is used in TPM2_HierarchyControl().

Table 63: Definition of (TPM_HANDLE) TPMI_RH_BASE_HIERARCHY Type

| Values | Comments |
|--------------------|---|
| TPM_RH_OWNER | Storage hierarchy |
| TPM_RH_PLATFORM | Platform hierarchy |
| TPM_RH_ENDORSEMENT | Endorsement hierarchy |
| #TPM_RC_VALUE | response code returned when the unmarshaling of this type fails |

9.18 TPMI_RH_PLATFORM

The Table 64 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 64: Definition of (TPM_HANDLE) TPMI_RH_PLATFORM Type

| Values | Comments |
|-----------------|---|
| TPM_RH_PLATFORM | Platform hierarchy |
| #TPM_RC_VALUE | response code returned when the unmarshaling of this type fails |

9.19 TPMI_RH_OWNER

The Table 65 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 65: Definition of (TPM_HANDLE) TPMI_RH_OWNER Type

| 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 |

9.20 TPMI_RH_ENDORSEMENT

The Table 66 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 Authorization is required.

Table 66: Definition of (TPM_HANDLE) TPMI_RH_ENDORSEMENT Type

| 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 |

9.21 TPMI_RH_PROVISION

The Table 67 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 67: Definition of (TPM_HANDLE) TPMSI_RH_PROVISION Type

| 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 |

9.22 TPMSI_RH_CLEAR

The Table 68 TPMSI_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 68: Definition of (TPM_HANDLE) TPMSI_RH_CLEAR Type

| 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 |

9.23 TPMSI_RH_NV_AUTH

The Table 69 interface type is used to identify the source of the authorization for access to an NV location. The handle value of a TPMSI_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 69: Definition of (TPM_HANDLE) TPMSI_RH_NV_AUTH Type

| 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 |

9.24 TPMSI_RH_LOCKOUT

The Table 70 TPMSI_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 70: Definition of (TPM_HANDLE) TPMI_RH_LOCKOUT Type

| Value | Comments |
|----------------|---|
| TPM_RH_LOCKOUT | handle for Lockout Authorization |
| #TPM_RC_VALUE | response code returned when the unmarshaling of this type fails |

9.25 TPMI_RH_NV_INDEX

The Table 71 interface type is used to identify an NV index. This type is used in all NV commands except those that define or undefine an index and TPM2_NV_ReadPublic().

Table 71: Definition of (TPM_HANDLE) TPMI_RH_NV_INDEX Type

| Value | Comments |
|--|--|
| {NV_INDEX_FIRST:NV_INDEX_LAST} | range of NV Indexes with 32-bit attributes |
| {EXTERNAL_NV_FIRST:EXTERNAL_NV_LAST} | range of external NV Indexes |
| {PERMANENT_NV_FIRST:PERMANENT_NV_LAST} | range of permanent NV Indexes |
| #TPM_RC_VALUE | error returned if the handle is out of range |

9.26 TPMI_RH_NV_DEFINED_INDEX

The Table 72 interface type is used to identify an NV index. This type is used in NV commands that define or undefine an index except for TPM2_NV_DefineSpace(). It does not apply to permanent NV Indexes, which are architecturally defined.

Table 72: Definition of (TPM_HANDLE) TPMI_RH_NV_DEFINED_INDEX Type

| Value | Comments |
|--------------------------------------|--|
| {NV_INDEX_FIRST:NV_INDEX_LAST} | range of NV Indexes with 32-bit attributes |
| {EXTERNAL_NV_FIRST:EXTERNAL_NV_LAST} | range of external NV Indexes |
| #TPM_RC_VALUE | error returned if the handle is out of range |

9.27 TPMI_RH_NV_LEGACY_INDEX

The Table 73 interface type is used to identify an NV index. This type is used in NV commands that use the TPM2_NV_DefineSpace() and TPM2_NV_ReadPublic() commands,

Table 73: Definition of (TPM_HANDLE) TPMI_RH_NV_LEGACY_INDEX Type

| Value | Comments |
|--------------------------------|--|
| {NV_INDEX_FIRST:NV_INDEX_LAST} | range of NV Indexes with 32-bit attributes |
| #TPM_RC_VALUE | error returned if the handle is out of range |

9.28 TPMI_RH_NV_EXP_INDEX

The Table 74 interface type is used to identify an NV index with expanded attributes. It is used in the TPMS_NV_PUBLIC_EXP_ATTR structure.

Table 74: Definition of (TPM_HANDLE) TPMI_RH_NV_EXP_INDEX Type

| Value | Comments |
|--------------------------------------|--|
| {EXTERNAL_NV_FIRST:EXTERNAL_NV_LAST} | range of external NV Indexes |
| #TPM_RC_VALUE | error returned if the handle is out of range |

9.29 TPMI_RH_AC

The Table 75 interface type is used to identify an attached component. This type is used in the AC commands.

Table 75: Definition of (TPM_HANDLE) TPMI_RH_AC Type

| Value | Comments |
|--------------------|--|
| {AC_FIRST:AC_LAST} | range of AC handles |
| #TPM_RC_VALUE | error returned if the handle is out of range |

9.30 TPMI_RH_ACT

The Table 76 interface type is used to identify the ACT instance used in TPM2_ACT_SetTimeout() and TPM2_SetPrimaryPolicy().

Table 76: 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 |

9.31 TPMI_ALG_HASH

Table 77 TPMI_ALG_HASH is an interface type of all the hash algorithms implemented on a specific TPM. The selector in Table 77 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 77: Definition of (TPM_ALG_ID) TPMI_ALG_HASH Type

| Values | Comments |
|----------------|----------|
| TPM_ALG_SHA1 | |
| TPM_ALG_SHA256 | |

(continued on next page)

(continued from previous page)

| Values | Comments |
|----------------------|----------|
| TPM_ALG_SHA384 | |
| TPM_ALG_SHA512 | |
| TPM_ALG_SHA256_192 | |
| TPM_ALG_SM3_256 | |
| TPM_ALG_SHA3_256 | |
| TPM_ALG_SHA3_384 | |
| TPM_ALG_SHA3_512 | |
| TPM_ALG_SHAKE256_192 | |
| TPM_ALG_SHAKE256_256 | |
| TPM_ALG_SHAKE256_512 | |
| +TPM_ALG_NULL | |
| #TPM_RC_HASH | |

9.32 TPMI_ALG_ASYM

This Table 78 TPMI_ALG_ASYM is an interface type of all the asymmetric algorithms implemented on a specific TPM. Table 78 lists each of the asymmetric algorithms that have an algorithm ID assigned by the TCG.

Table 78: Definition of (TPM_ALG_ID) TPMI_ALG_ASYM Type

| Values | Comments |
|--------------------|----------|
| TPM_ALG_RSA | |
| TPM_ALG_ECC | |
| +TPM_ALG_NULL | |
| #TPM_RC_ASYMMETRIC | |


9.33 TPMI_ALG_SYM

This Table 79 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 79: Definition of (TPM_ALG_ID) TPMI_ALG_SYM Type

| Values | Comments |
|-------------------|---|
| TPM_ALG_TDES | <p>Deprecated: TDES is deprecated. See Part 0.</p>  |
| TPM_ALG_AES | |
| TPM_ALG_XOR | required |
| TPM_ALG_SM4 | |
| TPM_ALG_CAMELLIA | |
| +TPM_ALG_NULL | required |
| #TPM_RC_SYMMETRIC | |


9.34 TPMI_ALG_SYM_OBJECT

This Table 80 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 80: Definition of (TPM_ALG_ID) TPMI_ALG_SYM_OBJECT Type

| Values | Comments |
|-------------------|---|
| TPM_ALG_TDES | <p>Deprecated: TDES is deprecated. See Part 0.</p>  |
| TPM_ALG_AES | |
| TPM_ALG_SM4 | |
| TPM_ALG_CAMELLIA | |
| +TPM_ALG_NULL | required |
| #TPM_RC_SYMMETRIC | |

9.35 TPMI_ALG_SYM_MODE

Table 81 TPMI_ALG_SYM_MODE is an interface type of all the TCG-defined block-cipher modes of operation.

Table 81: Definition of (TPM_ALG_ID) TPMI_ALG_SYM_MODE Type

| Values | Comments |
|---------------|----------|
| TPM_ALG_CTR | |
| TPM_ALG_OFB | |
| TPM_ALG_CBC | |
| TPM_ALG_CFB | |
| TPM_ALG_ECB | |
| TPM_ALG_CMAC | |
| +TPM_ALG_NULL | |
| #TPM_RC_MODE | |

9.36 TPMI_ALG_KDF

Table 82 TPMI_ALG_KDF is an interface type of all the key and mask derivation functions implemented on a specific TPM.

Table 82: Definition of (TPM_ALG_ID) TPMI_ALG_KDF Type

| Values | Comments |
|------------------------|----------|
| TPM_ALG_MGF1 | |
| TPM_ALG_KDF2 | |
| TPM_ALG_KDF1_SP800_56A | |
| TPM_ALG_KDF1_SP800_108 | |
| TPM_ALG_HKDF | |
| +TPM_ALG_NULL | |
| #TPM_RC_KDF | |

9.37 TPMI_ALG_SIG_SCHEME

Table 83 is the definition of the interface type for any signature scheme.

Table 83: Definition of (TPM_ALG_ID) TPMI_ALG_SIG_SCHEME Type

| Values | Comments |
|----------------|----------|
| TPM_ALG_RSASSA | |
| TPM_ALG_RSAPSS | |

(continued on next page)

(continued from previous page)

| Values | Comments |
|--------------------|--|
| TPM_ALG_ECDSA | |
| TPM_ALG_ECDA | |
| TPM_ALG_SM2 | |
| TPM_ALG_ECSCNORR | |
| TPM_ALG_EDDSA | |
| TPM_ALG_HASH_EDDSA | |
| TPM_ALG_HMAC | present in all TPM implementations |
| TPM_ALG_MLDSA | |
| TPM_ALG_HASH_MLDSA | |
| +TPM_ALG_NULL | |
| #TPM_RC_SCHEME | response code when a signature scheme is not correct |

9.38 TPMI_ECC_KEY_EXCHANGE

Table 84 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 are those that are dependent on ECC being implemented, even if they otherwise have the correct type attributes.

Table 84: Definition of (TPM_ALG_ID) (ECC) TPMI_ECC_KEY_EXCHANGE Type

| Values | Comments |
|----------------|---|
| TPM_ALG_ECDH | |
| TPM_ALG_SM2 | |
| TPM_ALG_ECMQV | |
| +TPM_ALG_NULL | |
| #TPM_RC_SCHEME | response code when a key exchange scheme is not correct |

9.39 TPMI_ST_COMMAND_TAG

The Table 85 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 85: Definition of (TPM_ST) TPMI_ST_COMMAND_TAG Type

| Values | Comments |
|--------------------|----------|
| TPM_ST_NO_SESSIONS | |
| TPM_ST_SESSIONS | |
| #TPM_RC_BAD_TAG | |

9.40 TPMI_ALG_MAC_SCHEME

Table 86 TPMI_ALG_MAC_SCHEME is an interface type of all the algorithm IDs that can be used with TPM2_MAC() and TPM2_MAC_Start().

Table 86: Definition of (TPM_ALG_ID) TPMI_ALG_MAC_SCHEME Type

| Values | Comments |
|----------------------|----------|
| TPM_ALG_CMAC | |
| TPM_ALG_SHA1 | |
| TPM_ALG_SHA256 | |
| TPM_ALG_SHA384 | |
| TPM_ALG_SHA512 | |
| TPM_ALG_SHA256_192 | |
| TPM_ALG_SM3_256 | |
| TPM_ALG_SHA3_256 | |
| TPM_ALG_SHA3_384 | |
| TPM_ALG_SHA3_512 | |
| TPM_ALG_SHAKE256_192 | |
| TPM_ALG_SHAKE256_256 | |
| TPM_ALG_SHAKE256_512 | |
| +TPM_ALG_NULL | |
| #TPM_RC_SYMMETRIC | |

9.41 TPMI_ALG_CIPHER_MODE

Table 87 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 87: Definition of (TPM_ALG_ID) TPMI_ALG_CIPHER_MODE Type

| Values | Comments |
|---------------|-----------------|
| TPM_ALG_CTR | |
| TPM_ALG_OFB | |
| TPM_ALG_CBC | |
| TPM_ALG_CFB | |
| TPM_ALG_ECB | |
| +TPM_ALG_NULL | required |
| #TPM_RC_MODE | |

10 Structure Definitions

10.1 TPMS_EMPTY

TPMS_EMPTY 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.

10.2 Hash/Digest Structures

10.2.1 TPMU_HA

Table 88 TPMU_HA is a union of all the hash algorithms implemented on a TPM.

Table 88: Definition of TPMU_HA Union

| Parameter | Type | Selector |
|-------------------|------|----------------------|
| sha1 [20] | BYTE | TPM_ALG_SHA1 |
| sha256 [32] | BYTE | TPM_ALG_SHA256 |
| sha384 [48] | BYTE | TPM_ALG_SHA384 |
| sha512 [64] | BYTE | TPM_ALG_SHA512 |
| sha256_192 [24] | BYTE | TPM_ALG_SHA256_192 |
| sm3_256 [32] | BYTE | TPM_ALG_SM3_256 |
| sha3_256 [32] | BYTE | TPM_ALG_SHA3_256 |
| sha3_384 [48] | BYTE | TPM_ALG_SHA3_384 |
| sha3_512 [64] | BYTE | TPM_ALG_SHA3_512 |
| shake256_192 [24] | BYTE | TPM_ALG_SHAKE256_192 |
| shake256_256 [32] | BYTE | TPM_ALG_SHAKE256_256 |
| shake256_512 [64] | BYTE | TPM_ALG_SHAKE256_512 |
| null | | TPM_ALG_NULL |

10.2.2 TPMT_HA

Table 89 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 89: Definition of TPMT_HA Structure

| Parameter | Type | Description |
|-----------|------|-------------|
|-----------|------|-------------|

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|------------------|----------------|--|
| hashAlg | +TPMI_ALG_HASH | selector of the hash contained in the <i>digest</i> that implies the size of the <i>digest</i> 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 |

10.3 Sized Buffers

10.3.1 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.

The high-order bit (0x8000) of *size* is a reserved bit, and shall be 0. That is, the maximum value of the *size* field in any TPM2B_ is 32767 (0x7FFF).

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:

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:

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:

In the Reference Code, 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.

10.3.2 TPM2B_DIGEST

The Table 90 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 90: Definition of TPM2B_DIGEST Structure

| Parameter | Type | Description |
|--------------------------------|--------|---|
| size | UINT16 | size in octets of the <i>buffer</i> field; may be 0 |
| buffer[size]{:sizeof(TPMU_HA)} | BYTE | the buffer area that can be no larger than a digest |

10.3.3 TPM2B_DATA

The Table 91 structure is used for a data buffer that is required to be no larger than the size of the Name of an object.

Table 91: Definition of TPM2B_DATA Structure

| Parameter | Type | Description |
|--------------------------------|--------|---|
| size | UINT16 | size in octets of the <i>buffer</i> field; may be 0 |
| buffer[size]{:sizeof(TPMT_HA)} | BYTE | |

10.3.4 TPM2B_NONCE

The Table 92 structure is used for a nonce.

Table 92: Definition of Types for TPM2B_NONCE

| Type | Name | Description |
|--------------|-------------|--|
| TPM2B_DIGEST | TPM2B_NONCE | size limited to the same as the digest structure |

10.3.5 TPM2B_AUTH

The Table 93 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 93: Definition of Types for TPM2B_AUTH

| Type | Name | Description |
|--------------|------------|--|
| TPM2B_DIGEST | TPM2B_AUTH | size limited to the same as the digest structure |

10.3.6 TPM2B_OPERAND

The Table 94 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 94: Definition of Types for TPM2B_OPERAND

| Type | Name | Description |
|--------------|---------------|--|
| TPM2B_DIGEST | TPM2B_OPERAND | size limited to the same as the digest structure |

10.3.7 TPM2B_EVENT

The Table 95 type is a sized buffer that can hold event data.

Table 95: Definition of TPM2B_EVENT Structure

| Parameter | Type | Description |
|-----------------------|--------|-----------------------------------|
| size | UINT16 | size of the operand <i>buffer</i> |
| buffer [size] {:1024} | BYTE | the event |

10.3.8 TPM2B_MAX_BUFFER

The Table 96 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(), TPM2_FieldUpgradeData(), or TPM2_SignSequenceComplete().

Note:

The above list is not comprehensive and other commands may use this buffer type.

Note:

MAX_2B_BUFFER_SIZE was MAX_DIGEST_SIZE prior to version 1.83, It is not related to a digest size. MAX_2B_BUFFER_SIZE is now just the maximum input data size to hash and HMAC commands.

MAX_2B_BUFFER_SIZE is TPM-dependent but is required to be at least 1,024.

Table 96: Definition of TPM2B_MAX_BUFFER Structure

| Parameter | Type | Description |
|--|--------|--------------------|
| size | UINT16 | size of the buffer |
| buffer [size] {:MAX_2B_BUFFER_SIZE} | BYTE | the buffer |

10.3.9 TPM2B_MAX_NV_BUFFER

The Table 97 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 97: Definition of TPM2B_MAX_NV_BUFFER Structure

| Parameter | Type | Description |
|--|--------|--|
| size | UINT16 | size of the buffer |
| buffer [size] {:MAX_NV_BUFFER_SIZE} | BYTE | the buffer Note: MAX_NV_BUFFER_SIZE is TPM-dependent |

10.3.10 TPM2B_TIMEOUT

The Table 98 TPM-dependent structure is used to provide the timeout value for an authorization. The *size* shall be 8 or less.

Table 98: 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 Code the MSb is used as a flag to indicate whether a ticket expires on TPM Reset or TPM Restart.

10.3.11 TPM2B_IV

The Table 99 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 99: Definition of TPM2B_IV Structure

| 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 |

10.3.12 TPM2B_SHARED_SECRET

The Table 100 type is a sized buffer that can hold the shared secret from a Key Encapsulation Mechanism. The size of this buffer depends on the KEM algorithm.

Table 100: Definition of TPM2B_SHARED_SECRET Structure

| Parameter | Type | Description |
|--|--------|--|
| size | UINT16 | size of the buffer |
| buffer [size] {:MAX_SHARED_SECRET_SIZE} | BYTE | the shared secret Note: Depending on the KEM, this shared secret may be suitable for use as a symmetric key, or may need to be processed by a Key Derivation Method (KDM) first. Note: MAX_SHARED_SECRET_SIZE is TPM-dependent and depends on the KEMs supported by the TPM. |

10.3.13 TPMU_KEM_CIPHERTEXT

A Table 101 TPMU_KEM_CIPHERTEXT holds a ciphertext associated with a KEM.

While the selector for this union is the KEM scheme associated with the key, note that every member of the union is a byte array, meaning that marshaling or unmarshaling code does not need to consider the selector when marshaling or unmarshaling a TPM2B_KEM_CIPHERTEXT.

Note:

Table 101 is illustrative. This structure is defined for the limited purpose of determining the size of a TPM2B_KEM_CIPHERTEXT. It would be modified depending on the algorithms supported in the TPM.

Table 101: Definition of TPMU_KEM_CIPHERTEXT Union

| Parameter | Type | Selector | Description |
|------------------------------|------|---------------|-------------|
| ecdh[sizeof(TPMS_ECC_POINT)] | BYTE | TPM_ALG_ECC | |
| mlkem[MAX_MLKEM_CT_SIZE] | BYTE | TPM_ALG_MLKEM | |

MAX_MLKEM_CT_SIZE is the maximum ciphertext size for any ML-KEM parameter set supported by the TPM. Table 205 lists the ciphertext sizes for each parameter set in FIPS 203 [4].

10.3.14 TPM2B_KEM_CIPHERTEXT

The Table 102 type is a sized buffer that can hold the ciphertext from a Key Encapsulation Mechanism. The size of this buffer depends on the KEM algorithm.

Table 102: Definition of TPM2B_KEM_CIPHERTEXT Structure

| Parameter | Type | Description |
|--|--------|--------------------|
| size | UINT16 | size of the buffer |
| buffer[size] {:sizeof(TPMU_KEM_CIPHERTEXT)} | BYTE | ciphertext |

10.3.15 TPM2B_VENDOR_PROPERTY

The Table 103 type is a sized buffer that can hold vendor property data.

Note:

This is an informative table that is included in this specification only to allow testing of the Reference Code. The vendor property is not required to be a TPM2B, nor is the 512 value normative.

Table 103: Definition of TPM2B_VENDOR_PROPERTY Structure

| Parameter | Type | Description |
|----------------------|--------|-----------------------------------|
| size | UINT16 | size of the operand <i>buffer</i> |
| buffer [size] {:512} | BYTE | the vendor property |

10.4 Names

10.4.1 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.

10.4.2 TPMU_NAME

Table 104 defines the TPMU_NAME union.

Table 104: 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 |

10.4.3 TPM2B_NAME

The Table 105 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 105: Definition of TPM2B_NAME Structure

| Parameter | Type | Description |
|--------------------------------|--------|----------------------------|
| size | UINT16 | size of the Name structure |
| name[size]{:sizeof(TPMU_NAME)} | BYTE | the Name structure |

10.5 PCR Structures

10.5.1 TPMS_PCR_SELECT

The Table 106 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:

The bit in *pcrSelect* corresponding to PCR 19 is in *pcrSelect* [2] ($19/8 = 2$).

The least significant bit in an octet is bit number 0. The bit in the octet associated with a PCR is the remainder after division by 8.

Example:

The bit in *pcrSelect* [2] corresponding to PCR 19 is bit 3 ($19 \bmod 8$). 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:

$$\text{PCR_SELECT_MIN} = (\text{PLATFORM_PCR} + 7) / 8 \quad (1)$$

where

PLATFORM_PCR is the number of PCR required by the platform-specific specification

The maximum value for *sizeofSelect* is:

$$\text{PCR_SELECT_MAX} = (\text{IMPLEMENTATION_PCR} + 7) / 8 \quad (2)$$

where

IMPLEMENTATION_PCR is 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:

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 106: Definition of TPMS_PCR_SELECT Structure

| Parameter | Type | Description |
|--|-------|--|
| sizeofSelect {PCR_SELECT_MIN:} | UINT8 | the size in octets of the <i>pcrSelect</i> array |
| pcrSelect [sizeofSelect] {;PCR_SELECT_MAX} | BYTE | the bit map of selected PCR |
| #TPM_RC_VALUE | | |

10.5.2 TPMS_PCR_SELECTION

Table 107 defines the TPMS_PCR_SELECTION Structure.

Table 107: 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 <i>pcrSelect</i> array |
| pcrSelect [sizeofSelect] {;PCR_SELECT_MAX} | BYTE | the bit map of selected PCR |

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|---------------|------|-------------|
| #TPM_RC_VALUE | | |

10.6 Tickets

10.6.1 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 Clause 10.6 are to be used by a TPM that is compliant with this specification.

The method of creating the ticket data is:

$$\text{HMAC}_{\text{contextAlg}}(\text{proof}, (\text{ticketType} \parallel \text{param} \{\parallel \text{param} \{\dots\}\})) \quad (3)$$

where

| | |
|-----------------------------------|--|
| $\text{HMAC}_{\text{contextAlg}}$ | is an HMAC using the hash used for context integrity |
| proof | is a TPM secret value (depends on hierarchy) |
| ticketType | is a value to differentiate the tickets |
| param | are one or more values that were checked by the TPM |

The proof value used for each hierarchy is shown in Table 108.

Table 108: 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 109. This is a template for the tickets shown in the remainder of Clause 10.6.

Table 109: General Format of a Ticket

| Parameter | Type | Description |
|-----------|--------|---|
| tag | TPM_ST | structure tag indicating the type of the ticket |

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|-----------|--------------------|--|
| hierarchy | TPMI_RH_HIERARCHY+ | the hierarchy of the proof value |
| digest | TPM2B_DIGEST | the HMAC over the ticket-specific data |

10.6.2 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.

10.6.3 TPMT_TK_CREATION

The Table 110 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

$$\text{HMAC}_{\text{contextAlg}}(\text{proof}, (\text{TPM_ST_CREATION} \parallel \text{name} \parallel \text{H}_{\text{nameAlg}}(\text{TPMS_CREATION_DATA}))) \quad (4)$$

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 110: Definition of TPMT_TK_CREATION Structure

| Parameter | Type | Description |
|-----------------------|--------------------|---|
| tag {TPM_ST_CREATION} | TPM_ST | ticket structure tag |
| #TPM_RC_TAG | | error returned when <i>tag</i> is not TPM_ST_CREATION |
| hierarchy | TPMI_RH_HIERARCHY+ | the hierarchy containing <i>name</i> |
| digest | TPM2B_DIGEST | This shall be the HMAC produced using a proof value of <i>hierarchy</i> . |

Example:

A NULL Creation Ticket is the tuple <TPM_ST_CREATION, TPM_RH_NULL, 0x0000>.

10.6.4 TPMU_TK_VERIFIED_META

The Table 111 TPMU_TK_VERIFIED_META contains additional metadata for a TPMT_TK_VERIFIED. Some types of TPMT_TK_VERIFIED (identified by their associated TPM_ST value) do not have any additional metadata.

Table 111: Definition of TPMU_TK_VERIFIED_META Union

| Parameter | Type | Selector | Description |
|-----------------|---------------|-------------------------|--|
| verified | TPMS_EMPTY | TPM_ST_VERIFIED | |
| messageVerified | TPMS_EMPTY | TPM_ST_MESSAGE_VERIFIED | |
| digestVerified | TPMI_ALG_HASH | TPM_ST_DIGEST_VERIFIED | The hash algorithm or XOF used to produce the digest that was verified |

10.6.5 TPMT_TK_VERIFIED

The Table 113 ticket is produced by TPM2_VerifySignature(), TPM2_VerifySequenceComplete(), and TPM2_VerifyDigestSignature(). It is used in TPM2_PolicyAuthorize().

The ticket provides evidence that the TPM has validated that a message or digest was signed by a key with the Name of *keyName*. The ticket contains an HMAC, which is computed by

$$\text{HMAC}_{\text{contextAlg}}(\text{proof}, (\text{tag} \parallel \text{digestOrMessage} \parallel \text{keyName} \parallel \text{metadata})) \quad (5)$$

where

| | |
|-----------------------------------|---|
| HMAC _{contextAlg} | an HMAC using the context integrity hash |
| proof | a TPM secret value associated with the hierarchy associated with <i>keyName</i> |
| tag | see Table 112 |
| digestOrMessage | the signed digest or message |
| keyName | Name of the key that signed <i>digestOrMessage</i> |
| metadata | Additional metadata depending on the type of ticket (may be empty-length) |

The *tag* value depends on which TPM command produced the ticket. Table 112 lists the values and associated commands.

The *metadata* is the serialized TPMU_TK_VERIFIED_META contents according to the *tag* value. For some cases, *metadata* is zero-length.

Table 112: TPM_ST Values For Signature Verification Commands

| <i>tag</i> value | Command |
|-------------------------|-------------------------------|
| TPM_ST_VERIFIED | TPM2_VerifySignature() |
| TPM_ST_MESSAGE_VERIFIED | TPM2_VerifySequenceComplete() |
| TPM_ST_DIGEST_VERIFIED | TPM2_VerifyDigestSignature() |

Table 113: Definition of TPMT_TK_VERIFIED Structure

| Parameter | Type | Description |
|---------------|-----------------------|---|
| tag | TPM_ST | ticket structure tag |
| #TPM_RC_TAG | | error returned when <i>tag</i> is not one of the values in Table 112 |
| hierarchy | TPMI_RH_HIERARCHY+ | the hierarchy containing <i>keyName</i> . |
| [tag]metadata | TPMU_TK_VERIFIED_META | Additional (if any) metadata required in order to validate the ticket. |
| hmac | TPM2B_DIGEST | This shall be the HMAC produced using a proof value of <i>hierarchy</i> . Note: Earlier versions of the TPM 2.0 specification called this field <i>digest</i> . It was renamed in version 185 to reduce ambiguity, since there are other digests associated with a TPMT_TK_VERIFIED. |

Example:

A NULL Verified Ticket is the tuple <*tag*, TPM_RH_NULL, 0x0000>, where *tag* is one of TPM_ST_VERIFIED, TPM_ST_MESSAGE_VERIFIED, or TPM_ST_DIGEST_VERIFIED.

10.6.6 TPMT_TK_AUTH

This Table 114 ticket is produced by TPM2_PolicySigned() and TPM2_PolicySecret() when the authorization has an expiration time. If *nonceTPM* was provided in the policy command, the ticket is computed by

$$\text{HMAC}_{\text{contextAlg}}(\text{proof}, (\text{TPM_ST_AUTH_xxx} \parallel \text{cpHash} \parallel \text{policyRef} \parallel \text{authName} \parallel \text{timeout} \parallel [\text{timeEpoch}] \parallel [\text{resetCount}])) \quad (6)$$

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:

This is not included if *timeout* is zero.

| | |
|------------|---|
| resetCount | implementation-specific representation of the TPM's totalResetCount |
|------------|---|

Note:

This is not included if *timeout* is zero or if *nonceTPM* was include in the authorization.

Table 114: 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 <i>tag</i> is not one of the allowed values |
| 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 <i>hierarchy</i> . |

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>

10.6.7 TPMT_TK_HASHCHECK

This Table 115 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

$$\text{HMAC}_{\text{contextAlg}}(\text{proof}, (\text{TPM_ST_HASHCHECK} \parallel \text{digest})) \quad (7)$$

where

| | |
|-----------------------------------|--|
| $\text{HMAC}_{\text{contextAlg}}$ | is an HMAC using the context integrity hash |
| proof | is a TPM secret value associated with the hierarchy indicated by the command |
| TPM_ST_HASHCHECK | is a value used to ensure that the ticket is properly used |
| digest | is the digest of the data |

Table 115: 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 <i>hierarchy</i> . |

10.7 Property Structures

10.7.1 TPMS_ALG_PROPERTY

This Table 116 structure is used to report the properties of an algorithm identifier. It is returned in response to a TPM2_GetCapability(*capability* == TPM_CAP_ALG).

Table 116: Definition of TPMS_ALG_PROPERTY Structure

| Parameter | Type | Description |
|---------------|----------------|---------------------------------|
| alg | TPM_ALG_ID | an algorithm identifier |
| algProperties | TPMA_ALGORITHM | the attributes of the algorithm |

10.7.2 TPMS_TAGGED_PROPERTY

This Table 117 structure is used to report the properties that are UINT32 values. It is returned in response to a TPM2_GetCapability().

Table 117: Definition of TPMS_TAGGED_PROPERTY Structure

| Parameter | Type | Description |
|-----------|--------|---------------------------|
| property | TPM_PT | a property identifier |
| value | UINT32 | the value of the property |

10.7.3 TPMS_TAGGED_PCR_SELECT

This Table 118 structure is used in TPM2_GetCapability() to return the attributes of the PCR.

Table 118: Definition of TPMS_TAGGED_PCR_SELECT Structure

| Parameter | Type | Description |
|---|------------|--|
| tag | TPM_PT_PCR | the property identifier |
| sizeofSelect {PCR_SELECT_MIN:} | UINT8 | the size in octets of the <i>pcrSelect</i> array |
| pcrSelect [sizeofSelect] {:PCR_SELECT_MAX} | BYTE | the bit map of PCR with the identified property |

10.7.4 TPMS_TAGGED_POLICY

This Table 119 structure is used in TPM2_GetCapability() to return the policy associated with a permanent handle.

Table 119: Definition of TPMS_TAGGED_POLICY Structure

| Parameter | Type | Description |
|------------|------------|-------------------------------|
| handle | TPM_HANDLE | a permanent handle |
| policyHash | TPMT_HA | the policy algorithm and hash |

10.7.5 TPMS_ACT_DATA

This Table 120 structure is used in TPM2_GetCapability() to return the ACT data.

Table 120: Definition of TPMS_ACT_DATA Structure

| 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 |

10.7.6 TPMS_SPDM_SESSION_INFO

This Table 121 structure is only used in TPM2_GetCapability(*capability* == TPM_CAP_SPDM_SESSION_INFO) and returns the SPDM session information associated with the current SPDM session (i.e., the session in which TPM2_GetCapability() is sent).

Table 121: Definition of TPMS_SPDM_SESSION_INFO Structure

| Parameter | Type | Description |
|------------|------------|---|
| reqKeyName | TPM2B_NAME | Name of the Requester key associated with the current SPDM session An Empty Buffer is allowed and indicates that the SPDM session was established with Responder-only authentication |
| tpmKeyName | TPM2B_NAME | Name of the TPM key associated with the current SPDM session |

10.8 Lists

10.8.1 TPML_CC

The Table 122 list of command codes may be input to the TPM or returned by the TPM depending on the command.

Table 122: Definition of TPML_CC Structure

| Parameter | Type | Description |
|----------------------------------|--------|---|
| count | UINT32 | number of commands in the <i>commandCode</i> 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 |

10.8.2 TPML_CCA

This Table 123 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 43) with vendor-specific commands returned after other commands. Because of the other attributes, the commands may not be returned in strict numerical order.

Table 123: Definition of TPML_CCA Structure

| Parameter | Type | Description |
|---------------------------------------|---------|---|
| count | UINT32 | number of values in the <i>commandAttributes</i> list; may be 0 |
| commandAttributes[count]{:MAX_CAP_CC} | TPMA_CC | a list of command codes attributes |

10.8.3 TPML_ALG

This Table 124 list is returned by TPM2_IncrementalSelfTest().

Table 124: Definition of TPML_ALG Structure

| Parameter | Type | Description |
|---------------------------------------|------------|---|
| count | UINT32 | number of algorithms in the <i>algorithms</i> 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 <i>count</i> is greater than the maximum allowed list size |

10.8.4 TPML_HANDLE

This Table 125 structure is used when the TPM returns a list of loaded handles in response to TPM2_GetCapability(*capability* == TPM_CAP_HANDLE).

Note:

```
MAX_CAP_HANDLES = (MAX_CAP_DATA / sizeof(TPM_HANDLE))
```

Table 125: Definition of TPML_HANDLE Structure

| 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 <i>count</i> is greater than the maximum allowed list size |

10.8.5 TPML_DIGEST

This Table 126 list is used to convey a list of digest values. This type is used in TPM2_PolicyOR() and in TPM2_PCR_Read().

Table 126: Definition of TPML_DIGEST Structure

| Parameter | Type | Description |
|------------|--------|--|
| count {2:} | UINT32 | number of digests in the list The minimum is two for TPM2_PolicyOR(). |

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|--------------------|--------------|--|
| 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 <i>count</i> is not at least two or is greater than eight |

10.8.6 TPML_DIGEST_VALUES

This Table 127 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:

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:

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 127: 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 <i>count</i> is greater than the possible number of banks |

10.8.7 TPML_PCR_SELECTION

This Table 128 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 128: 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 <i>count</i> is greater than the possible number of banks |

10.8.8 TPML_ALG_PROPERTY

This Table 129 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 129: Definition of TPML_ALG_PROPERTY Structure

| 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 |

10.8.9 TPML_TAGGED_TPM_PROPERTY

This Table 130 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 130: Definition of TPML_TAGGED_TPM_PROPERTY Structure

| 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 |

10.8.10 TPML_TAGGED_PCR_PROPERTY

This Table 131 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 131: Definition of TPML_TAGGED_PCR_PROPERTY Structure

| 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 |

10.8.11 TPML_ECC_CURVE

This Table 132 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 132: Definition of (ECC) TPML_ECC_CURVE Structure

| 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 |

10.8.12 TPML_TAGGED_POLICY

This Table 133 list is used to report the authorization policy values for permanent handles. This is list may be generated by TPM2_GetCapability(). 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 133: Definition of TPML_TAGGED_POLICY Structure

| 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 |

10.8.13 TPML_ACT_DATA

This Table 134 list is used to report the timeout and state for the ACT. This list may be generated by TPM2_GetCapability(). Only implemented ACT are present in the list

Note:

MAX_ACT_DATA = MAX_CAP_DATA / sizeof(TPMS_ACT_DATA).

Table 134: Definition of TPML_ACT_DATA Structure

| 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 |

10.8.14 TPML_PUB_KEY

This Table 135 list is only used in TPM2_GetCapability(capability == TPM_CAP_PUB_KEYS).

Note:

MAX_PUB_KEYS = MAX_CAP_DATA / sizeof(TPM2B_PUBLIC).

Table 135: Definition of TPML_PUB_KEY Structure

| Parameter | Type | Description |
|-------------------------------|--------------|--|
| count | UINT32 | number of public keys A value of zero is allowed. |
| pubKeys[count]{:MAX_PUB_KEYS} | TPM2B_PUBLIC | array of public keys |

10.8.15 TPML_SPDM_SESSION_INFO

This Table 136 list is only used in TPM2_GetCapability(capability == TPM_CAP_SPDM_SESSION_INFO) and returns the SPDM session information associated with the current SPDM session (i.e., the session in which TPM2_GetCapability() is sent). If TPM2_GetCapability() is not sent within an SPDM session, an empty list will be returned.

Note:

MAX_SPDM_SESSION_INFO = MAX_CAP_DATA / sizeof(TPMS_SPDM_SESSION_INFO).

Table 136: Definition of TPML_SPDM_SESSION_INFO Structure

| Parameter | Type | Description |
|-----------|------|-------------|
|-----------|------|-------------|

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|--|------------------------|--|
| count | UINT32 | number of SPDM session information structures A value of zero is allowed. |
| spdmSessionInfo[count] {:MAX_SPDM_SESSION_INFO} | TPMS_SPDM_SESSION_INFO | array of SPDM session information |

10.8.16 TPML_VENDOR_PROPERTY

This Table 137 list is used to convey a list of vendor specific properties.

Note:

$MAX_VENDOR_PROPERTY = MAX_CAP_DATA / \text{sizeof}(TPM2B_VENDOR_PROPERTY)$.

Note:

This is an informative table that is included in the specification only to allow testing of the Reference Code.

Table 137: Definition of TPML_VENDOR_PROPERTY Structure

| Parameter | Type | Description |
|---|-----------------------|--|
| count | UINT32 | number of vendor properties in the list A value of zero is allowed. |
| vendorData[count] {:MAX_VENDOR_PROPERTY} | TPM2B_VENDOR_PROPERTY | a list of vendor properties |

10.9 Capabilities Structures

It is required that each parameter in the union described in Table 138 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:

$$MAX_CAP_DATA = MAX_CAP_BUFFER - \text{sizeof}(TPM_CAP) - \text{sizeof}(UINT32)$$

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.

Note:

TPML_VENDOR_PROPERTY was added in version 1.83. Older TPMs can return a single TPMS_TAGGED_PROPERTY without a count.

10.9.1 TPMU_CAPABILITIES

Table 138: Definition of TPMU_CAPABILITIES Union

| Parameter | Type | Selector |
|-----------------|--------------------------|---------------------------|
| 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 |
| authPolicies | TPML_TAGGED_POLICY | TPM_CAP_AUTH_POLICIES |
| actData | TPML_ACT_DATA | TPM_CAP_ACT |
| pubKeys | TPML_PUB_KEY | TPM_CAP_PUB_KEYS |
| spdmSessionInfo | TPML_SPDM_SESSION_INFO | TPM_CAP_SPDM_SESSION_INFO |
| vendorProperty | TPML_VENDOR_PROPERTY | TPM_CAP_VENDOR_PROPERTY |

10.9.2 TPMS_CAPABILITY_DATA

This Table 139 data area is returned in response to a TPM2_GetCapability().

Table 139: Definition of TPMS_CAPABILITY_DATA Structure

| Parameter | Type | Description |
|------------------|-------------------|---------------------|
| capability | TPM_CAP | the capability |
| [capability]data | TPMU_CAPABILITIES | the capability data |

10.9.3 TPMS_SET_CAPABILITY_DATA

This Table 140 structure is used in TPM2_SetCapability().

Table 140: Definition of TPMS_SET_CAPABILITY_DATA Structure

| Parameter | Type | Description |
|---------------------|-----------------------|-------------------------------|
| setCapability | TPM_CAP | the capability to be set |
| [setCapability]data | TPMU_SET_CAPABILITIES | the capability data to be set |

Note:

The TPMU_SET_CAPABILITIES Structure may be defined by a TCG Registry.

10.9.4 TPM2B_SET_CAPABILITY_DATA

This Table 141 structure is used in TPM2_SetCapability().

Table 141: Definition of TPM2B_SET_CAPABILITY_DATA Structure

| Parameter | Type | Description |
|-------------------|--|-------------------------------|
| size= | UINT16 | size of setCapabilityData |
| setCapabilityData | TPMS_SET_CAPABILITY_DATA | the capability data to be set |

10.10 Clock/Counter Structures

10.10.1 TPMS_CLOCK_INFO

This Table 142 structure is used in each of the attestation commands.

Table 142: 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 (<i>clock</i> =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 <i>Clock</i> greater than the current value of <i>Clock</i> has been previously reported by the TPM. Set to YES on TPM2_Clear(). |

10.10.2 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 2^{22} milliseconds (~69.9 minutes). The update rate of the non-volatile portion of *Clock* shall be reported by a `TPM2_GetCapability(capability == TPM_CAP_TPM_PROPERTIES, property == TPM_PT_CLOCK_UPDATE)`.

10.10.3 *ResetCount*

This counter shall increment on each TPM Reset. This counter shall be reset to zero by `TPM2_Clear()`.

10.10.4 *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()`.

10.10.5 *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:

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()`.

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()`.

10.10.6 `TPMS_TIME_INFO`

This Table 143 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 143: Definition of `TPMS_TIME_INFO` Structure

| Parameter | Type | Description |
|-----------|------------------------------|---|
| time | UINT64 | time in milliseconds since the <i>Time</i> circuit was last reset This structure element is used to report on the TPM's <i>Time</i> value. |
| clockInfo | <code>TPMS_CLOCK_INFO</code> | a structure containing the clock information |

10.11 TPM Attestation Structures

10.11.1 Introduction

Clause 10.11 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.

10.11.2 TPMS_TIME_ATTEST_INFO

This Table 144 structure is used when the TPM performs TPM2_GetTime().

Table 144: Definition of TPMS_TIME_ATTEST_INFO Structure

| Parameter | Type | Description |
|-----------------|----------------|--|
| time | TPMS_TIME_INFO | the <i>Time</i> , <i>Clock</i> , <i>resetCount</i> , <i>restartCount</i> , and <i>Safe</i> indicator |
| firmwareVersion | UINT64 | a TPM vendor-specific value indicating the version number of the firmware |

10.11.3 TPMS_CERTIFY_INFO

This Table 145 structure is the attested data for TPM2_Certify().

Table 145: Definition of TPMS_CERTIFY_INFO Structure

| Parameter | Type | Description |
|---------------|------------|--|
| name | TPM2B_NAME | Name of the certified object |
| qualifiedName | TPM2B_NAME | Qualified Name of the certified object |

10.11.4 TPMS_QUOTE_INFO

This Table 146 structure is the *attested* data for TPM2_Quote().

Table 146: Definition of TPMS_QUOTE_INFO Structure

| Parameter | Type | Description |
|-----------|--------------------|--|
| pcrSelect | TPML_PCR_SELECTION | information on <i>algID</i> , PCR selected and digest |
| pcrDigest | TPM2B_DIGEST | digest of the selected PCR using the hash of the signing key |

10.11.5 TPMS_COMMAND_AUDIT_INFO

This Table 147 structure is the *attested* data for TPM2_GetCommandAuditDigest().

Table 147: Definition of TPMS_COMMAND_AUDIT_INFO Structure

| 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 <i>digestAlg</i> |

10.11.6 TPMS_SESSION_AUDIT_INFO

This Table 148 structure is the *attested* data for TPM2_GetSessionAuditDigest().

Table 148: Definition of TPMS_SESSION_AUDIT_INFO Structure

| Parameter | Type | Description |
|------------------|--------------|--|
| exclusiveSession | TPMI_YES_NO | current exclusive status of the session YES if all of the commands recorded in the <i>sessionDigest</i> 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 |

10.11.7 TPMS_CREATION_INFO

This Table 149 structure is the *attested* data for TPM2_CertifyCreation().

Table 149: Definition of TPMS_CREATION_INFO Structure

| Parameter | Type | Description |
|--------------|--------------|--------------------|
| objectName | TPM2B_NAME | Name of the object |
| creationHash | TPM2B_DIGEST | creationHash |

10.11.8 TPMS_NV_CERTIFY_INFO

This Table 150 structure contains the Name and contents of the selected NV Index that is certified by TPM2_NV_Certify().

Table 150: Definition of TPMS_NV_CERTIFY_INFO Structure

| Parameter | Type | Description |
|------------|---------------------|--|
| indexName | TPM2B_NAME | Name of the NV Index |
| offset | UINT16 | the <i>offset</i> parameter of TPM2_NV_Certify() |
| nvContents | TPM2B_MAX_NV_BUFFER | contents of the NV Index |

10.11.9 TPMS_NV_DIGEST_CERTIFY_INFO

This Table 151 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 version 1.59 to support alternate TPM2_NV_Certify() behavior.

Table 151: Definition of TPMS_NV_DIGEST_CERTIFY_INFO Structure

| Parameter | Type | Description |
|-----------|--------------|-----------------------------------|
| indexName | TPM2B_NAME | Name of the NV Index |
| nvDigest | TPM2B_DIGEST | hash of the contents of the index |

10.11.10 TPMI_ST_ATTEST

This Table 152 is the selector for the TPMU_ATTEST.

Table 152: Definition of (TPM_ST) TPMI_ST_ATTEST Type

| 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() |

10.11.11 TPMU_ATTEST

Table 153 defines the TPMU_ATTEST Structure.

Table 153: Definition of TPMU_ATTEST Union

| 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 |

10.11.12 TPMS_ATTEST

This Table 154 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 154: Definition of TPMS_ATTEST Structure

| Parameter | Type | Description |
|-----------------|-----------------|---|
| magic | TPM_CONSTANTS32 | 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 |

10.11.13 TPM2B_ATTEST

This Table 155 sized buffer contains the signed structure. The *attestationData* is the signed portion of the structure. The *size* parameter is not signed.

Table 155: Definition of TPM2B_ATTEST Structure

| Parameter | Type | Description |
|---|--------|--|
| size | UINT16 | size of the <i>attestationData</i> structure |
| attestationData[size]{{:sizeof(TPMS_ATTEST)}} | BYTE | the signed structure |

10.12 Authorization Structures

10.12.1 Introduction

The structures in Clause 10.12 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.

10.12.2 TPMS_AUTH_COMMAND

This Table 155 structure is the format used for each of the authorizations in the session area of a command.

Table 156: Definition of TPMS_AUTH_COMMAND Structure

| 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 |

10.12.3 TPMS_AUTH_RESPONSE

This Table 157 structure 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 157: Definition of TPMS_AUTH_RESPONSE Structure

| 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 |

11 Algorithm Parameters and Structures

11.1 Symmetric

11.1.1 Introduction

This clause defines the parameters and structures for describing symmetric algorithms.


11.1.2 Symmetric Cipher Key Sizes

This Table 158 interface type defines the supported key sizes for each symmetric algorithm. This type is used to allow an unmarshaling routine to generate the proper validation code (TPM_RC_VALUE) for an unsupported key sizes. An implementation that supports different key sizes would have a different set of selections.

Note:

Key size is expressed in bits.

Table 158: Definition of Types for Symmetric Cipher Key Sizes

| Type | Name | Allowed Values |
|--------------|---|-----------------|
| TPM_KEY_BITS | TPMI_TDES_KEY_BITS <i>Deprecated:</i> TDES is deprecated. See Part 0.  | {128, 192} |
| TPM_KEY_BITS | TPMI_AES_KEY_BITS | {128, 192, 256} |
| TPM_KEY_BITS | TPMI_SM4_KEY_BITS | {128} |
| TPM_KEY_BITS | TPMI_CAMELLIA_KEY_BITS | {128, 192, 256} |


11.1.3 TPMU_SYM_KEY_BITS

This Table 159 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 159: Definition of TPMU_SYM_KEY_BITS Union

| Parameter | Type | Selector | Description |
|-----------|--------------------|--------------|---|
| tDES | TPMI_TDES_KEY_BITS | TPM_ALG_TDES | <i>Deprecated:</i> TDES is deprecated. See Part 0.  |

(continued on next page)

(continued from previous page)

| Parameter | Type | Selector | Description |
|-----------|------------------------|------------------|---|
| aes | TPMI_AES_KEY_BITS | TPM_ALG_AES | |
| xor | TPMI_ALG_HASH | TPM_ALG_XOR | overload for using <i>xor</i> Note: TPM_ALG_NULL is not allowed. |
| sm4 | TPMI_SM4_KEY_BITS | TPM_ALG_SM4 | |
| camellia | TPMI_CAMELLIA_KEY_BITS | TPM_ALG_CAMELLIA | |
| null | | TPM_ALG_NULL | |


11.1.4 TPMU_SYM_MODE

Table 160 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 160: Definition of TPMU_SYM_MODE Union

| Parameter | Type | Selector | Description |
|-----------|--------------------|------------------|---|
| tDES | TPMI_ALG_SYM_MODE+ | TPM_ALG_TDES | Deprecated: TDES is deprecated. See Part 0.  |
| aes | TPMI_ALG_SYM_MODE+ | TPM_ALG_AES | |
| xor | TPMS_EMPTY | TPM_ALG_XOR | no mode selector |
| sm4 | TPMI_ALG_SYM_MODE+ | TPM_ALG_SM4 | |
| camellia | TPMI_ALG_SYM_MODE+ | TPM_ALG_CAMELLIA | |
| null | TPMS_EMPTY | TPM_ALG_NULL | no mode selector |

11.1.5 TPMU_SYM_DETAILS

This Table 161 union allows additional parameters to be added for a symmetric cipher. Currently, no additional parameters are required for any of the symmetric algorithms.

Table 161: Definition of TPMU_SYM_DETAILS Union

| Parameter | Type | Selector | Description |
|-----------|------------|------------------|--|
| tdes | TPMS_EMPTY | TPM_ALG_TDES | <p>Deprecated:</p> <p>TDES is deprecated. See Part 0.</p>  |
| aes | TPMS_EMPTY | TPM_ALG_AES | |
| xor | TPMS_EMPTY | TPM_ALG_XOR | no mode selector |
| sm4 | TPMS_EMPTY | TPM_ALG_SM4 | |
| camellia | TPMS_EMPTY | TPM_ALG_CAMELLIA | |
| null | TPMS_EMPTY | TPM_ALG_NULL | no mode selector |

11.1.6 TPMT_SYM_DEF

The Table 162 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 162: 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, if any |

11.1.7 TPMT_SYM_DEF_OBJECT

This Table 163 structure is used when different symmetric block cipher (not XOR) algorithms may be selected.

Table 163: Definition of TPMT_SYM_DEF_OBJECT Structure

| Parameter | Type | Description |
|--------------------|----------------------|--|
| algorithm | +TPMI_ALG_SYM_OBJECT | <p>selects a symmetric block cipher</p> <p>When used in the parameter area of a parent object, this shall be a supported block cipher and not TPM_ALG_NULL</p> |
| [algorithm]keyBits | TPMU_SYM_KEY_BITS | the key size |

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|--------------------|------------------|--|
| [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 |

11.1.8 TPM2B_SYM_KEY

This Table 164 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 164: 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 |

11.1.9 TPMS_SYMCIPHER_PARMS

This Table 165 structure contains the parameters for a symmetric block cipher object.

Table 165: Definition of TPMS_SYMCIPHER_PARMS Structure

| Parameter | Type | Description |
|-----------|---------------------|--------------------------|
| sym | TPMT_SYM_DEF_OBJECT | a symmetric block cipher |

11.1.10 TPM2B_LABEL

This Table 166 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 166: Definition of TPM2B_LABEL Structure

| Parameter | Type | Description |
|---------------------------------|--------|---|
| size | UINT16 | |
| buffer[size]{:LABEL_MAX_BUFFER} | BYTE | symmetric data for a created object or the <i>label</i> and <i>context</i> for a derived object |

11.1.11 TPMS_DERIVE

This Table 167 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 *inPublic* area template take precedence over the values in the *inSensitive* parameter.

Table 167: Definition of TPMS_DERIVE Structure

| Parameter | Type | Description |
|-----------|-------------|-------------|
| label | TPM2B_LABEL | |
| context | TPM2B_LABEL | |

11.1.12 TPM2B_DERIVE

Table 168 structure defines the TPM2B_DERIVE Structure.

Table 168: Definition of TPM2B_DERIVE Structure

| Parameter | Type | Description |
|-------------------------------------|--------|---|
| size | UINT16 | |
| buffer[size]{: sizeof(TPMS_DERIVE)} | BYTE | symmetric data for a created object or the <i>label</i> and <i>context</i> for a derived object |

11.1.13 TPMU_SENSITIVE_CREATE

This Table 169 structure allows a TPM2B_SENSITIVE_CREATE structure to carry either a TPM2B_SENSITIVE_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.

Table 169: Definition of TPMU_SENSITIVE_CREATE Union

| Parameter | Type | Selector | Description |
|----------------------|------|----------|---|
| create[MAX_SYM_DATA] | BYTE | | sensitive data for a created symmetric Object |

(continued on next page)

(continued from previous page)

| Parameter | Type | Selector | Description |
|-----------|-------------|----------|--|
| derive | TPMS_DERIVE | | <i>label</i> and <i>context</i> for a derived Object |

11.1.14 TPM2B_SENSITIVE_DATA

This Table 170 buffer wraps the TPMU_SENSITIVE_CREATE Structure.

Table 170: 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 <i>label</i> and <i>context</i> for a derived object |

11.1.15 TPMS_SENSITIVE_CREATE

This Table 171 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* of *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 171: Definition of TPMS_SENSITIVE_CREATE Structure

| Parameter | Type | Description |
|-----------|----------------------|--|
| userAuth | TPM2B_AUTH | the USER auth secret value |
| data | TPM2B_SENSITIVE_DATA | data to be sealed, a key, or derivation values |

11.1.16 TPM2B_SENSITIVE_CREATE

This Table 172 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 172: Definition of TPM2B_SENSITIVE_CREATE Structure

| Parameter | Type | Description |
|-----------|------|-------------|
|-----------|------|-------------|

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|-----------|-----------------------|--|
| size= | UINT16 | size of <i>sensitive</i> in octets (may not be zero) <div style="background-color: #f0f0f0; padding: 5px;"> <p>Note: The <i>userAuth</i> and data parameters in this buffer can 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.</p> </div> |
| sensitive | TPMS_SENSITIVE_CREATE | data to be sealed or a symmetric key value. |

11.1.17 TPMS_SCHEME_HASH

This Table 173 structure is the scheme data for schemes that only require a hash to complete their definition.

Table 173: Definition of TPMS_SCHEME_HASH Structure

| Parameter | Type | Description |
|-----------|---------------|---|
| hashAlg | TPMI_ALG_HASH | the hash algorithm used to digest the message |

11.1.18 TPMS_SCHEME_ECDA

This Table 174 definition is for split signing schemes that require a commit count.

Table 174: Definition of (ECC) TPMS_SCHEME_ECDA 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 |

11.1.19 TPMI_ALG_KEYEDHASH_SCHEME

This Table 175 is the list of values that may appear in a *keyedHash* as the *scheme* parameter.

Table 175: 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 | |

11.1.20 TPMS_SCHEME_HMAC

This Table 176 is the list of TPMS_SCHEME_HMAC values.

Table 176: Definition of Types for TPMS_SCHEME_HMAC

| Type | Name | Description |
|------------------|------------------|-------------|
| TPMS_SCHEME_HASH | TPMS_SCHEME_HMAC | |

11.1.21 TPMS_SCHEME_XOR

This Table 177 structure is for the XOR encryption scheme.

Note:

Prior to version 1.59, 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 177: 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 |

11.1.22 TPMU_SCHEME_KEYEDHASH

Table 178 defines the TPMU_SCHEME_KEYEDHASH union.

Table 178: Definition of TPMU_SCHEME_KEYEDHASH Union

| 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 | |

11.1.23 TPMT_KEYEDHASH_SCHEME

This Table 179 structure is used for a hash signing object.

Table 179: 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 |

11.2 Asymmetric

11.2.1 Signing Schemes

11.2.1.1 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.

11.2.1.2 RSA Signature Schemes

These are the Table 180 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 180: Definition of (RSA) Types for RSA Signature Schemes

| Type | Name |
|------------------|------------------------|
| TPMS_SCHEME_HASH | TPMS_SIG_SCHEME_RSASSA |
| TPMS_SCHEME_HASH | TPMS_SIG_SCHEME_RSAPSS |

11.2.1.3 ECC Signature Schemes

Most of the Table 181 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_ECDA.

Table 181: Definition of (ECC) Types for ECC Signature Schemes

| Type | Name |
|------------------|----------------------------|
| TPMS_SCHEME_HASH | TPMS_SIG_SCHEME_ECDSA |
| TPMS_SCHEME_ECDA | TPMS_SIG_SCHEME_ECDA |
| TPMS_SCHEME_HASH | TPMS_SIG_SCHEME_SM2 |
| TPMS_SCHEME_HASH | TPMS_SIG_SCHEME_EC Schnorr |
| TPMS_EMPTY | TPMS_SIG_SCHEME_EDDSA |
| TPMS_EMPTY | TPMS_SIG_SCHEME_HASH_EDDSA |

Note:

The SM2 Digital Signature Algorithm does not specify a specific cryptographic hash algorithm. The TPM does not automatically provide a default; if TPMS_SIG_SCHEME_SM2.hashAlg is TPM_ALG_NULL, the TPM will return an error. However, per section 5.4.2 of [19], this value should usually be SM3 (TPM_ALG_SM3_256).

11.2.1.4 TPMU_SIG_SCHEME

This Table 182 is the union of all of the signature schemes.

Table 182: Definition of TPMU_SIG_SCHEME Union

| Parameter | Type | Selector |
|------------|----------------------------|--------------------|
| rsassa | TPMS_SIG_SCHEME_RSASSA | TPM_ALG_RSASSA |
| rsapss | TPMS_SIG_SCHEME_RSAPSS | TPM_ALG_RSAPSS |
| ecdsa | TPMS_SIG_SCHEME_ECDSA | TPM_ALG_ECDSA |
| ecdaa | TPMS_SIG_SCHEME_ECDA | TPM_ALG_ECDA |
| sm2 | TPMS_SIG_SCHEME_SM2 | TPM_ALG_SM2 |
| ecschnorr | TPMS_SIG_SCHEME_ECSCHNORR | TPM_ALG_ECSCHNORR |
| eddsa | TPMS_SIG_SCHEME_EDDSA | TPM_ALG_EDDSA |
| hash_eddsa | TPMS_SIG_SCHEME_HASH_EDDSA | TPM_ALG_HASH_EDDSA |
| hmac | TPMS_SCHEME_HMAC | TPM_ALG_HMAC |
| null | | TPM_ALG_NULL |

11.2.1.5 TPMT_SIG_SCHEME

Table 183 defines TPMT_SIG_SCHEME.

Table 183: Definition of TPMT_SIG_SCHEME Structure

| Parameter | Type | Description |
|-----------------|----------------------|-------------------|
| scheme | +TPMI_ALG_SIG_SCHEME | scheme selector |
| [scheme]details | TPMU_SIG_SCHEME | scheme parameters |

11.2.2 Encryption Schemes

11.2.2.1 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.

11.2.2.2 RSA Encryption Schemes

Table 184 contains the RSA encryption schemes.

Table 184: Definition of Types for RSA Encryption Schemes

| Type | Name |
|------------------|-----------------------|
| TPMS_EMPTY | TPMS_ENC_SCHEME_RSAES |
| TPMS_SCHEME_HASH | TPMS_ENC_SCHEME_OAEP |

11.2.2.3 ECC Key Exchange Schemes

Table 185 contains the ECC key exchange schemes.

Table 185: Definition of Types for (ECC) ECC Key Exchange

| Type | Name |
|------------------|-----------------------|
| TPMS_SCHEME_HASH | TPMS_KEY_SCHEME_ECDH |
| TPMS_SCHEME_HASH | TPMS_KEY_SCHEME_SM2 |
| TPMS_SCHEME_HASH | TPMS_KEY_SCHEME_ECMQV |

11.2.3 Key Derivation Schemes

11.2.3.1 Introduction

These Table 186 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 186: Definition of Types for KDF Schemes

| Type | Name |
|------------------|--------------------------------|
| TPMS_SCHEME_HASH | TPMS_KDF_SCHEME_MGF1 |
| TPMS_SCHEME_HASH | TPMS_KDF_SCHEME_KDF2 |
| TPMS_SCHEME_HASH | TPMS_KDF_SCHEME_KDF1_SP800_56A |
| TPMS_SCHEME_HASH | TPMS_KDF_SCHEME_KDF1_SP800_108 |
| TPMS_SCHEME_HASH | TPMS_KDF_SCHEME_HKDF |

11.2.3.2 TPMU_KDF_SCHEME

Table 187 defines TPMU_KDF_SCHEME.

Table 187: Definition of TPMU_KDF_SCHEME Union

| Parameter | Type | Selector |
|----------------|--------------------------------|------------------------|
| mgf1 | TPMS_KDF_SCHEME_MGF1 | TPM_ALG_MGF1 |
| kdf2 | TPMS_KDF_SCHEME_KDF2 | TPM_ALG_KDF2 |
| kdf1_sp800_56a | TPMS_KDF_SCHEME_KDF1_SP800_56A | TPM_ALG_KDF1_SP800_56A |
| kdf1_sp800_108 | TPMS_KDF_SCHEME_KDF1_SP800_108 | TPM_ALG_KDF1_SP800_108 |
| hkdf | TPMS_KDF_SCHEME_HKDF | TPM_ALG_HKDF |
| null | | TPM_ALG_NULL |

11.2.3.3 TPMT_KDF_SCHEME

Table 188 defines TPMT_KDF_SCHEME.

Table 188: Definition of TPMT_KDF_SCHEME Structure

| Parameter | Type | Description |
|-----------------|-----------------|-------------------|
| scheme | +TPMI_ALG_KDF | scheme selector |
| [scheme]details | TPMU_KDF_SCHEME | scheme parameters |

11.2.4 RSA

11.2.4.1 TPMI_ALG_RSA_SCHEME

The Table 189 list of values that may appear in the scheme parameter of a TPMS_RSA_PARMS Structure.

Table 189: Definition of (TPM_ALG_ID) (RSA) TPMI_ALG_RSA_SCHEME Type

| Values | Comments |
|----------------|----------|
| TPM_ALG_RSASSA | |
| TPM_ALG_RSAES | |
| TPM_ALG_RSAPSS | |
| TPM_ALG_OAEP | |
| +TPM_ALG_NULL | |
| #TPM_RC_VALUE | |

11.2.4.2 TPMU_RSA_SCHEME

Table 190 defines the TPMU_RSA_SCHEME.

Table 190: Definition of (RSA) TPMU_RSA_SCHEME Union

| Parameter | Type | Selector | Description |
|-----------|------------------|----------------|--------------|
| rsassa | TPMS_SCHEME_HASH | TPM_ALG_RSASSA | |
| rsaes | TPMS_EMPTY | TPM_ALG_RSAES | |
| rsapss | TPMS_SCHEME_HASH | TPM_ALG_RSAPSS | |
| oaep | TPMS_SCHEME_HASH | TPM_ALG_OAEP | |
| null | | TPM_ALG_NULL | unpadded RSA |

11.2.4.3 TPMT_RSA_SCHEME

Table 191 defines the TPMT_RSA_SCHEME.

Table 191: Definition of (RSA) TPMT_RSA_SCHEME Structure

| Parameter | Type | Description |
|-----------------|----------------------|-------------------|
| scheme | +TPMI_ALG_RSA_SCHEME | scheme selector |
| [scheme]details | TPMU_RSA_SCHEME | scheme parameters |

11.2.4.4 TPMI_ALG_RSA_DECRYPT

The Table 192 list of values that are allowed in a decryption scheme selection as used in TPM2_RSA_Encrypt() and TPM2_RSA_Decrypt().

Table 192: Definition of (TPM_ALG_ID) (RSA) TPMI_ALG_RSA_DECRYPT Type

| Values | Comments |
|---------------|----------|
| TPM_ALG_RSAES | |
| TPM_ALG_OAEP | |
| +TPM_ALG_NULL | |
| #TPM_RC_VALUE | |

11.2.4.5 TPMT_RSA_DECRYPT

Table 193 defines TPMT_RSA_DECRYPT.

Table 193: Definition of (RSA) TPMT_RSA_DECRYPT Structure

| Parameter | Type | Description |
|-----------------|-----------------------|-------------------|
| scheme | +TPMI_ALG_RSA_DECRYPT | scheme selector |
| [scheme]details | TPMU_RSA_SCHEME | scheme parameters |

11.2.4.6 TPM2B_PUBLIC_KEY_RSA

This Table 194 sized buffer holds the largest RSA public key supported by the TPM.

Table 194: 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 |

11.2.4.7 TPMI_RSA_KEY_BITS

This Table 195 holds the value that is the maximum size allowed for an RSA key.

Note:

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.

Table 195: 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 |

11.2.4.8 TPM2B_PRIVATE_KEY_RSA

This Table 196 sized buffer holds the largest RSA prime number supported by the TPM.

Note:

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:

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 version 1.59.

Table 196: Definition of (RSA) TPM2B_PRIVATE_KEY_RSA Structure

| Parameter | Type | Description |
|-----------|------|-------------|
|-----------|------|-------------|

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|----------------------------------|--------|-------------|
| size | UINT16 | |
| buffer[size]{:RSA_PRIVATE_SIZE } | BYTE | |

11.2.5 ECC

11.2.5.1 TPM2B_ECC_PARAMETER

This Table 197 sized buffer holds the largest ECC parameter (coordinate) supported by the TPM.

Table 197: Definition of (ECC) TPM2B_ECC_PARAMETER Structure

| Parameter | Type | Description |
|--------------------------------------|--------|-----------------------|
| size | UINT16 | size of <i>buffer</i> |
| buffer[size] {:MAX_ECC_KEY_BYTES} | BYTE | the parameter data |

11.2.5.2 TPMS_ECC_POINT

This Table 198 structure holds two ECC coordinates that, together, make up an ECC point.

y is zero-length for the Edwards curves (Curve25519 and Curve448). See Part 1 for details on how ECC points are represented using this structure.

Table 198: Definition of (ECC) TPMS_ECC_POINT Structure

| Parameter | Type | Description |
|-----------|-------------------------------------|--------------|
| x | TPM2B_ECC_PARAMETER | X coordinate |
| y | TPM2B_ECC_PARAMETER | Y coordinate |

11.2.5.3 TPM2B_ECC_POINT

This Table 199 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 199: Definition of (ECC) TPM2B_ECC_POINT Structure

| Parameter | Type | Description |
|-----------|--------------------------------|---|
| size= | UINT16 | size of the remainder of this structure |
| point | TPMS_ECC_POINT | coordinates |

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|--------------|------|--|
| #TPM_RC_SIZE | | error returned if the unmarshaled size of <i>point</i> is not exactly equal to <i>size</i> |

11.2.5.4 TPMI_ALG_ECC_SCHEME

Table 200 defines TPMI_ALG_ECC_SCHEME.

Table 200: Definition of (TPM_ALG_ID) (ECC) TPMI_ALG_ECC_SCHEME Type

| Values | Comments |
|--------------------|----------|
| TPM_ALG_ECDSA | |
| TPM_ALG_ECDA | |
| TPM_ALG_SM2 | |
| TPM_ALG_ECSCNORR | |
| TPM_ALG_EDDSA | |
| TPM_ALG_HASH_EDDSA | |
| TPM_ALG_ECDH | |
| TPM_ALG_SM2 | |
| TPM_ALG_ECMQV | |
| +TPM_ALG_NULL | |
| #TPM_RC_SCHEME | |

11.2.5.5 TPMI_ECC_CURVE

Table 201 enumerates the ECC curves implemented by the TPM.

Numeric values of TPM_ECC_x correspond to the numeric values of TCG_ECC_x from the TCG Algorithm Registry [1].

Table 201: Definition of (ECC) (TPM_ECC_CURVE) TPMI_ECC_CURVE Type

| Parameter | Value | Comment |
|-------------------|--------|---------|
| +TPM_ECC_NONE | 0x0000 | |
| TPM_ECC_NIST_P192 | 0x0001 | |
| TPM_ECC_NIST_P224 | 0x0002 | |
| TPM_ECC_NIST_P256 | 0x0003 | |

(continued on next page)

(continued from previous page)

| Parameter | Value | Comment |
|---------------------|-----------------------------------|------------------------|
| 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_BP_P256_R1 | 0x0030 | Brainpool |
| TPM_ECC_BP_P384_R1 | 0x0031 | Brainpool |
| TPM_ECC_BP_P512_R1 | 0x0032 | Brainpool |
| TPM_ECC_CURVE_25519 | 0x0040 | curve to support EdDSA |
| TPM_ECC_CURVE_448 | 0x0041 | curve to support EdDSA |
| #TPM_RC_CURVE | error when curve is not supported | |

11.2.5.6 TPMU_ECC_SCHEME

Table 202 defines the TPMU_ECC_SCHEME.

Table 202: Definition of (ECC) TPMU_ECC_SCHEME Union

| Parameter | Type | Selector | Description |
|------------|-------------------|--------------------|-------------|
| ecdsa | TPMS_SCHEME_HASH | TPM_ALG_ECDSA | |
| ecschnorr | TPMS_SCHEME_HASH | TPM_ALG_ECSCHNORR | |
| eddsa | TPMS_EMPTY | TPM_ALG_EDDSA | |
| hash_eddsa | TPMS_EMPTY | TPM_ALG_HASH_EDDSA | |
| ecdac | TPMS_SCHEME_ECDAA | TPM_ALG_ECDAA | |

11.2.5.7 TPMT_ECC_SCHEME

Table 203 defines TPMT_ECC_SCHEME.


Table 203: Definition of (TPMT_SIG_SCHEME) (ECC) TPMT_ECC_SCHEME Structure

| Parameter | Type | Description |
|-----------------|----------------------|-------------------|
| scheme | +TPMI_ALG_ECC_SCHEME | scheme selector |
| [scheme]details | TPMU_ECC_SCHEME | scheme parameters |

11.2.5.8 TPMS_ALGORITHM_DETAIL_ECC

This Table 204 structure is used to report on the curve parameters of an ECC curve. It is returned by TPM2_ECC_Parameters().

Table 204: Definition of (ECC) TPMS_ALGORITHM_DETAIL_ECC Structure

| Parameter | Type | Description |
|-----------|---------------------|--|
| curveID | TPM_ECC_CURVE | identifier for the curve |
| keySize | UINT16 | size in bits of the key |
| kdf | TPMT_KDF_SCHEME+ | the default KDF and hash algorithm recommended for use in secret sharing operations (may be TPM_ALG_NULL) <div style="background-color: #f0f0f0; padding: 5px; border: 1px solid #ccc;"> <p>Note: The TPM does not use or check this value. It is provided as an informative recommendation to users.</p> </div> <div style="background-color: #f0f0f0; padding: 5px; border: 1px solid #ccc; margin-top: 10px;"> <p>Deprecated: This field was deprecated in version 185. See Part 0. </p> </div> |
| 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 | F_p (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 | co-factor (a size of zero indicates a co-factor of 1) |

11.2.6 ML-KEM

11.2.6.1 TPMI_MLKEM_PARMS

Table 205 enumerates the ML-KEM parameter sets that can be implemented by a TPM.

Numeric values of TPM_MLKEM_x correspond to the numeric values of TCG_MLKEM_x from the TCG Algorithm Registry [1].

Public key, ciphertext, and shared secret key sizes correspond to the listed sizes for each parameter set in FIPS 203 [4].

Table 205: Definition of (MLKEM) TPMI_MLKEM_PARMS Type

| Parameter set | Numeric value | Public key size in bytes | Ciphertext size in bytes | Shared secret key size in bytes |
|---------------|---------------|--------------------------|--------------------------|---------------------------------|
| TPM_MLKEM_512 | 0x0001 | 800 | 768 | 32 |

(continued on next page)

(continued from previous page)

| Parameter set | Numeric value | Public key size in bytes | Ciphertext size in bytes | Shared secret key size in bytes |
|----------------|---|--------------------------|--------------------------|---------------------------------|
| TPM_MLKEM_768 | 0x0002 | 1184 | 1088 | 32 |
| TPM_MLKEM_1024 | 0x0003 | 1568 | 1568 | 32 |
| #TPM_RC_PARAMS | error when parameter set is not supported | | | |

11.2.6.2 TPM2B_PUBLIC_KEY_MLKEM

Table 206 defines the TPM2B_PUBLIC_KEY_MLKEM structure.

A TPM2B_PUBLIC_KEY_MLKEM contains an encoded ML-KEM public key, according to Algorithm 19 (ML-KEM.KeyGen) of FIPS 203 [4].

Table 206: Definition of (MLKEM) TPM2B_PUBLIC_KEY_MLKEM Structure

| Parameter | Type | Description |
|--|--------|--------------------|
| size | UINT16 | size of the buffer |
| buffer [size] {:MAX_MLKEM_PUB_SIZE} | BYTE | the buffer |

MAX_MLKEM_PUB_SIZE is the maximum public key size for any ML-KEM parameter set supported by the TPM.

Table 205 lists the public key sizes for each parameter set in FIPS 203 [4].

11.2.6.3 TPM2B_PRIVATE_KEY_MLKEM

Table 207 defines the TPM2B_PRIVATE_KEY_MLKEM structure.

Note:

Table 207 defines the canonical TPM wire format for ML-KEM private keys. As an implementation detail, a TPM can cache the expanded private key in TPM2B_PRIVATE_VENDOR_SPECIFIC rather than re-generating it every time it is needed.

Table 207: Definition of (MLKEM) TPM2B_PRIVATE_KEY_MLKEM Structure

| Parameter | Type | Description |
|------------|--------|-------------------------------------|
| size | UINT16 | shall be 64 |
| buffer[64] | BYTE | the 64-byte private seed ($d z$) |

11.2.7 ML-DSA

11.2.7.1 TPMI_MLDSA_PARAMS

Table 208 enumerates the ML-DSA parameter sets that can be implemented by a TPM.

Numeric values of TPM_MLDSA_x correspond to the numeric values of TCG_MLDSA_x from the TCG Algorithm Registry [1].

Public key and signature sizes correspond to the listed sizes for each parameter set in FIPS 204 [5].

Table 208: Definition of (MLDSA) TPMS_MLDSA_PARMS Type

| Parameter set | Numeric value | Public key size in bytes | Signature size in bytes |
|---------------|---|--------------------------|-------------------------|
| TPM_MLDSA_44 | 0x0001 | 1312 | 2420 |
| TPM_MLDSA_65 | 0x0002 | 1952 | 3309 |
| TPM_MLDSA_87 | 0x0003 | 2592 | 4627 |
| #TPM_RC_PARMS | error when parameter set is not supported | | |

11.2.7.2 TPMS_SIGNATURE_HASH_MLDSA

Table 209 defines the TPMS_SIGNATURE_HASH_MLDSA structure.

Table 209: Definition of (PHMLDSA) TPMS_SIGNATURE_HASH_MLDSA Structure

| Parameter | Type | Description |
|-----------|-----------------------|---|
| hash | TPMI_ALG_HASH | the hash algorithm used to digest the message TPM_ALG_NULL is not allowed. |
| signature | TPM2B_SIGNATURE_MLDSA | |

11.2.7.3 TPM2B_PUBLIC_KEY_MLDSA

Table 210 defines the TPM2B_PUBLIC_KEY_MLDSA structure.

A TPM2B_PUBLIC_KEY_MLDSA contains an encoded ML-DSA public key, according to Algorithm 22 (pkEncode) of FIPS 204 [5].

Table 210: Definition of (MLDSA) TPM2B_PUBLIC_KEY_MLDSA Structure

| Parameter | Type | Description |
|--|--------|--------------------|
| size | UINT16 | size of the buffer |
| buffer [size] {:MAX_MLDSA_PUB_SIZE} | BYTE | the buffer |

MAX_MLDSA_PUB_SIZE is the maximum public key size for any ML-DSA parameter set supported by the TPM.

Table 208 lists the public key sizes for each parameter set in FIPS 204 [5].

11.2.7.4 TPM2B_PRIVATE_KEY_MLDSA

Table 211 defines the TPM2B_PRIVATE_KEY_MLDSA structure.

Note:

Table 211 defines the canonical TPM wire format for ML-DSA private keys. As an implementation detail, a TPM can cache the expanded private key in TPM2B_PRIVATE_VENDOR_SPECIFIC rather than re-generating it every time it is needed.

Table 211: Definition of (MLDSA) TPM2B_PRIVATE_KEY_MLDSA Structure

| Parameter | Type | Description |
|------------|--------|--------------------------------|
| size | UINT16 | shall be 32 |
| buffer[32] | BYTE | the 32-byte private seed ξ |

11.3 Signatures

11.3.1 TPMS_SIGNATURE_RSA

Table 212 defines the TPMS_SIGNATURE_RSA Structure. Table 213 defines the types.

Table 212: 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 213: Definition of Types for (RSA) Signature

| Type | Name |
|--------------------|-----------------------|
| TPMS_SIGNATURE_RSA | TPMS_SIGNATURE_RSASSA |
| TPMS_SIGNATURE_RSA | TPMS_SIGNATURE_RSAPSS |

11.3.2 TPMS_SIGNATURE_ECC

Table 214 defines the TPMS_SIGNATURE_ECC Structure. Table 215 defines the types.

Table 214: 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 215: Definition of Types for (ECC) TPMS_SIGNATURE_ECC

| Type | Name |
|------|------|
|------|------|

(continued on next page)

(continued from previous page)

| Type | Name |
|--------------------|--------------------------|
| TPMS_SIGNATURE_ECC | TPMS_SIGNATURE_ECDSA |
| TPMS_SIGNATURE_ECC | TPMS_SIGNATURE_ECDA |
| TPMS_SIGNATURE_ECC | TPMS_SIGNATURE_SM2 |
| TPMS_SIGNATURE_ECC | TPMS_SIGNATURE_ECSCHNORR |

11.3.3 TPM2B_SIGNATURE_EDDSA

Table 216 defines the TPM2B_SIGNATURE_EDDSA Structure.

Table 216: Definition of (ECC) TPM2B_SIGNATURE_EDDSA Structure

| Parameter | Type | Description |
|---------------------------------------|--------|---|
| size | UINT16 | size in octets of the <i>buffer</i> field; may be 0 |
| buffer[size]{ :2 * MAX_ECC_KEY_BYTES} | BYTE | the buffer area that can be no larger than twice the size of an ECC parameter |

11.3.4 TPM2B_SIGNATURE_MLDSA

Table 217 defines the TPM2B_SIGNATURE_MLDSA structure.

Table 217: Definition of (MLDSA) TPM2B_SIGNATURE_MLDSA Structure

| Parameter | Type | Description |
|--------------------------------------|--------|--------------------|
| size | UINT16 | size of the buffer |
| buffer [size] { :MAX_MLDSA_SIG_SIZE} | BYTE | the buffer |

MAX_MLDSA_SIG_SIZE is the maximum signature size for any ML-DSA parameter set supported by the TPM.

Table 208 lists the signature sizes for each parameter set in FIPS 204 [5].

11.3.5 TPMU_SIGNATURE

A Table 218 TPMU_SIGNATURE 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 TPMs 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 218: Definition of TPMU_SIGNATURE Union

| Parameter | Type | Selector |
|------------|---------------------------|--------------------|
| rsassa | TPMS_SIGNATURE_RSASSA | TPM_ALG_RSASSA |
| rsapss | TPMS_SIGNATURE_RSAPSS | TPM_ALG_RSAPSS |
| ecdsa | TPMS_SIGNATURE_ECDSA | TPM_ALG_ECDSA |
| ecdaa | TPMS_SIGNATURE_ECDA | TPM_ALG_ECDA |
| sm2 | TPMS_SIGNATURE_SM2 | TPM_ALG_SM2 |
| ecschnorr | TPMS_SIGNATURE_ECSCHNORR | TPM_ALG_ECSCHNORR |
| eddsa | TPM2B_SIGNATURE_EDDSA | TPM_ALG_EDDSA |
| hash_eddsa | TPM2B_SIGNATURE_EDDSA | TPM_ALG_HASH_EDDSA |
| mldsa | TPM2B_SIGNATURE_MLDSA | TPM_ALG_MLDSA |
| hash_mlds | TPMS_SIGNATURE_HASH_MLDSA | TPM_ALG_HASH_MLDSA |
| hmac | TPMT_HA | TPM_ALG_HMAC |
| null | | TPM_ALG_NULL |

Note:

Table 218's *mldsa*, *eddsa*, and *hash_eddsa* members are TPM2B types instead of TPMS types, because unlike the other signature types, there is no choice of hash algorithm to include in the signature's metadata.

11.3.6 TPMT_SIGNATURE

Table 219 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, `TPM2_SignSequenceComplete()`, and `TPM2_SignDigest()`. It is an input to commands such as `TPM2_VerifySequenceComplete()`, `TPM2_VerifyDigestSignature()`, `TPM2_PolicySigned()`, and `TPM2_FieldUpgradeStart()`.

Table 219: 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. |

11.3.7 TPMU_SIGNATURE_CTX

A Table 220 TPMU_SIGNATURE_CTX allows a signature scheme to include additional contextual information.

While the selector for this union is the signing scheme associated with the key, note that every member of the union is a byte array, meaning that marshaling or unmarshaling code does not need to consider the selector when marshaling or unmarshaling a TPM2B_SIGNATURE_CTX.

Note:

Table 220 is illustrative. This structure is defined for the limited purpose of determining the size of a TPM2B_SIGNATURE_CTX.

Table 220: Definition of TPMU_SIGNATURE_CTX Union

| Parameter | Type | Selector(s) | Description |
|-----------------------------|------|-------------------------------------|--|
| commitCount[sizeof(UINT16)] | BYTE | TPM_ALG_ECDSA | the counter value that is used between TPM2_Commit() and the sign operation |
| id[MAX_SM2_ID_BYTES] | BYTE | TPM_ALG_SM2 | Value for ID_A “identifying an entity’s identity” |
| buffer[255] | BYTE | TPM_ALG_MLDSA or TPM_ALG_HASH_MLDSA | optional context value used by the algorithm |
| empty[0] | BYTE | all other values | all other signature schemes do not support additional context Note: This is defined as a zero-length buffer so that all members of TPMU_SIGNATURE_CTX have the same basic type. |

11.3.8 TPM2B_SIGNATURE_CTX

This Table 221 sized buffer contains additional signature context in the form of a TPMU_SIGNATURE_CTX.

The *context* buffer contains an appropriate value for the type of key being used with the signature or verification operation (e.g., TPM2_SignSequenceComplete or TPM2_VerifySequenceComplete).

Table 221: Definition of TPM2B_SIGNATURE_CTX Structure

| Parameter | Type | Description |
|--|--------|------------------------|
| size | UINT16 | size of <i>context</i> |
| context[size] {:sizeof(TPMU_SIGNATURE_CTX)} | BYTE | |

11.3.9 TPM2B_SIGNATURE_HINT

This Table 222 sized buffer contains an algorithm-specific signature hint to be provided in TPM2_VerifySequenceStart().

- For TPM_ALG_EDDSA, *hint* contains the encoded R value from the signature.

- For all other signature algorithms, this buffer must be zero-length.

MAX_SIGNATURE_HINT_SIZE is an implementation-dependent value that is the maximum hint size across all signature verification algorithms supported by the TPM.

Table 222: Definition of TPM2B_SIGNATURE_HINT Structure

| Parameter | Type | Description |
|---------------------------------------|--------|---------------------|
| size | UINT16 | size of <i>hint</i> |
| hint[size] {:MAX_SIGNATURE_HINT_SIZE} | BYTE | |

11.4 Key/Secret Exchange

11.4.1 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.

11.4.2 TPMU_ENCRYPTED_SECRET

This Table 223 structure is used to hold either a Labeled KEM ciphertext (see Part 1, “Labeled Key Encapsulation Mechanism”), 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 223 is illustrative. It would be modified depending on the algorithms supported in the TPM.

Table 223: 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 | |
| mlkem[MAX_MLKEM_CT_SIZE] | BYTE | TPM_ALG_MLKEM | |
| symmetric[sizeof(TPM2B_DIGEST)] | BYTE | TPM_ALG_SYMCIPHER | |

(continued on next page)

(continued from previous page)

| Parameter | Type | Selector | Description |
|---|------|-------------------|--|
| keyedHash[sizeof(TPM2B_DIGEST)] | BYTE | TPM_ALG_KEYEDHASH | any symmetrically encrypted secret value's maximum size will be limited to the digest size of the associated algorithm |

11.4.3 TPM2B_ENCRYPTED_SECRET

Table [224](#) defines the TPM2B_ENCRYPTED_SECRET Structure.

Table 224: Definition of TPM2B_ENCRYPTED_SECRET Structure

| Parameter | Type | Description |
|--|--------|--------------------------|
| size | UINT16 | size of the secret value |
| secret[size] {:sizeof(TPMU_ENCRYPTED_SECRET)} | BYTE | secret |

12 Key/Object Complex

12.1 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.

12.2 Public Area Structures

12.2.1 Description

This clause 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.

12.2.2 TPMI_ALG_PUBLIC

Table 225 defines the TPMI_ALG_PUBLIC type.

Table 225: Definition of (TPM_ALG_ID) TPMI_ALG_PUBLIC Type

| Values | Comments |
|--------------------|---|
| TPM_ALG_RSA | |
| TPM_ALG_KEYEDHASH | |
| TPM_ALG_ECC | |
| TPM_ALG_SYMCIPHER | |
| TPM_ALG_MLDSA | |
| TPM_ALG_HASH_MLDSA | |
| TPM_ALG_MLKEM | |
| #TPM_RC_TYPE | response code when a public type is not supported |

12.2.3 Type-Specific Parameters

12.2.3.1 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:

$$unique = H_{nameAlg}(seedValue \parallel sensitive) \quad (8)$$

where

| | |
|------------------|--|
| $H_{nameAlg}$ | is the hash algorithm used to compute the Name of the object |
| <i>seedValue</i> | is the digest-sized obfuscation value in the sensitive area of a symmetric key or symmetric data object found in a TPMT_SENSITIVE.seedValue.buffer |
| <i>sensitive</i> | is the secret key/data of the object in the TPMT_SENSITIVE.sensitive.any.buffer |

12.2.3.2 TPMU_PUBLIC_ID

This Table 226 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 226: Definition of TPMU_PUBLIC_ID Union

| 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 | |
| mldsa | TPM2B_PUBLIC_KEY_MLDSA | TPM_ALG_MLDSA or TPM_ALG_HASH_MLDSA | |
| mlkem | TPM2B_PUBLIC_KEY_MLKEM | TPM_ALG_MLKEM | |
| derive | TPMS_DERIVE | | only allowed for TPM2_CreateLoaded() when <i>parentHandle</i> is a Derivation Parent. |


12.2.3.3 TPMS_KEYEDHASH_PARMS

This Table 227 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 227: Definition of TPMS_KEYEDHASH_PARMS Structure

| Parameter | Type | Description |
|-----------|------------------------|---|
| scheme | TPMT_KEYEDHASH_SCHEME+ | <p>indicates the signing method used for a <i>keyedHash</i> signing object</p> <p>This field also determines the size of the data field for a data object created with TPM2_Create() or TPM2_CreatePrimary().</p> <div style="background-color: #f0f0f0; padding: 5px;"> <p>Deprecated:</p> <p>Support for TPM_ALG_NULL for HMAC keys with the <i>sign</i> attribute was deprecated in version 185. See Part 0.</p>  </div> |

12.2.3.4 TPMS_RSA_PARMS

Table 228 defines the TPMS_RSA_PARMS Structure.

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 $2^{16} + 1$. 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 or generate the key if *exponent* is not zero.

Table 228: Definition of (RSA) TPMS_RSA_PARMS Structure

| Parameter | Type | Description |
|-----------|----------------------|---|
| symmetric | TPMT_SYM_DEF_OBJECT+ | <p>For a restricted decryption key, shall be set to a supported symmetric algorithm, key size, and mode.</p> <p>If the key is not a restricted decryption key, this field shall be set to TPM_ALG_NULL.</p> |

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|-----------|-------------------|--|
| scheme | TPMT_RSA_SCHEME+ | <p>scheme.scheme shall be:</p> <p>for an unrestricted signing key, either TPM_ALG_RSAPSS TPM_ALG_RSASSA or TPM_ALG_NULL</p> <p>for a restricted signing key, either TPM_ALG_RSAPSS or TPM_ALG_RSASSA</p> <p>for an unrestricted decryption key, TPM_ALG_RSAES, TPM_ALG_OAEP, or TPM_ALG_NULL unless the object also has the <i>sign</i> attribute</p> <p>for a restricted decryption key, TPM_ALG_NULL</p> <p>When both sign and decrypt are SET, restricted shall be CLEAR and scheme shall be TPM_ALG_NULL.</p> <div style="background-color: #f0f0f0; padding: 5px;"> <p>Deprecated:  Support for TPM_ALG_NULL except for Storage Keys, and keys intended for use with the raw RSAEP/RSADP primitive, was deprecated in version 185. See Part 0.</p> </div> |
| keyBits | TPMI_RSA_KEY_BITS | number of bits in the public modulus |
| exponent | UINT32 | <p>the public exponent</p> <p>an odd number greater than 2.</p> |

12.2.3.5 TPMS_ECC_PARMS


This Table 229 structure contains the parameters for prime modulus ECC.

Table 229: Definition of (ECC) TPMS_ECC_PARMS Structure

| Parameter | Type | Description |
|-----------|----------------------|---|
| symmetric | TPMT_SYM_DEF_OBJECT+ | <p>For a restricted decryption key, shall be set to a supported symmetric algorithm, key size. and mode.</p> <p>if the key is not a restricted decryption key, this field shall be set to TPM_ALG_NULL.</p> <p>The TPM will return TPM_RC_SYMMETRIC if any of the above requirements are not met.</p> |

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|-----------|------------------|---|
| scheme | TPMT_ECC_SCHEME+ | <p>If the <i>sign</i> attribute of the key is SET, then this shall be a valid signing scheme.</p> <p>If the <i>sign</i> parameter in <i>curveID</i> indicates a mandatory scheme, then this field shall have the same value.</p> <p>If the <i>decrypt</i> attribute of the key is SET, then this shall be a valid key exchange scheme or TPM_ALG_NULL.</p> <p>If the key is a Storage Key, then this field shall be TPM_ALG_NULL.</p> <p>If the key is not a Storage Key, and <i>curveID</i> is either TPM_ECC_CURVE_25519 or TPM_ECC_CURVE_448, then this field shall not be TPM_ALG_NULL.</p> <p>The TPM will return TPM_RC_SCHEME if any of the above requirements are not met.</p> <div style="background-color: #f0f0f0; padding: 5px;"><p>Deprecated:</p><p>Support for TPM_ALG_NULL except for Storage Keys was deprecated in version 185. See Part 0.</p></div> |
| curveID | TPMI_ECC_CURVE | ECC curve ID |

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|-----------|------------------|--|
| kdf | TPMT_KDF_SCHEME+ | <p>if the key is an unrestricted decryption TPM_ALG_ECDH key, an optional key derivation scheme</p> <p>Shall be NULL in all other cases (TPM_RC_KDF).</p> <p>If this field is not NULL, then this key can be used with TPM2_Encapsulate() and TPM2_Decapsulate() (as well as TPM2_ECDH_KeyGen() and TPM2_ECDH_ZGen()). In the context of TPM2_Encapsulate() and TPM2_Decapsulate(), the KEM is equivalent to DHKEM(<i>curveID</i>, <i>kdf</i>) from RFC 9180 [25]</p> <p>Note:</p> <p>Currently, TPM_ALG_HKDF is the only supported KDF for DHKEM.</p> <p>In the context of object creation, TPM2_LoadExternal(), or TPM2_TestParms(), TPM_RC_KDF indicates the TPM does not support the requested KDF.</p> <p>Note:</p> <p>In the context of TPM2_Encapsulate() and TPM2_Decapsulate(), <i>scheme.details.ecdh.hashAlg</i> is ignored, because <i>kdf</i> specifies all parameters of the KDF, including any hash algorithm.</p> <p>If this field is NULL, then this key cannot be used with TPM2_Encapsulate() and TPM2_Decapsulate().</p> |

12.2.3.6 TPMS_MLDSA_PARMS

Table 230 contains the parameters for ML-DSA keys.

Table 230: Definition of (MLDSA) TPMS_MLDSA_PARMS Structure

| Parameter | Type | Description |
|--------------|------------------|-------------------------|
| parameterSet | TPMI_MLDSA_PARMS | ML-DSA parameter set ID |

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|-----------------|-------------|---|
| allowExternalMu | TPMI_YES_NO | <p>If YES, this key can be used with TPM2_VerifyDigestSignature() and TPM2_SignDigest(). In the context of these two commands, the <i>digest</i> value will be interpreted as the 512-byte external Mu (μ) value as computed in Algorithm 7 (ML-DSA.Sign_internal), Line 6 of FIPS 204 [5].</p> <p>If NO, this key cannot be used with TPM2_VerifyDigestSignature() and TPM2_SignDigest().</p> <p>If a TPM supports ML-DSA but not external Mu, it will return TPM_RC_EXT_MU in object creation and TPM2_TestParms() if <i>allowExternalMu</i> is YES.</p> <div style="background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <p>Note:</p> <p>ML-DSA keys can always be used with TPM2_SignSequenceComplete and TPM2_VerifySequenceComplete().</p> </div> |

12.2.3.7 TPMS_HASH_MLDSA_PARMS

Table 231 contains the parameters for Pre-Hash ML-DSA keys.

Table 231: Definition of (PHMLDSA) TPMS_HASH_MLDSA_PARMS Structure

| Parameter | Type | Description |
|--------------|------------------|--------------------------|
| parameterSet | TPMI_MLDSA_PARMS | ML-DSA parameter set ID |
| hashAlg | TPMI_ALG_HASH | the pre-hash function PH |

12.2.3.8 TPMS_MLKEM_PARMS

Table 232 defines the parameters structure for ML-KEM keys.

Table 232: Definition of (MLKEM) TPMS_MLKEM_PARMS Structure

| Parameter | Type | Description |
|--------------|----------------------|---|
| symmetric | TPMT_SYM_DEF_OBJECT+ | <p>For a restricted decryption key, shall be set to a supported symmetric algorithm, key size. and mode.</p> <p>if the key is not a restricted decryption key, this field shall be set to TPM_ALG_NULL.</p> |
| parameterSet | TPMI_MLKEM_PARMS | The ML-KEM parameter set |

12.2.3.9 TPMU_PUBLIC_PARMS

Table 233 defines the possible parameter definition structures that may be contained in the public portion of a key.

Note:

All object types that can be parents will have a [TPMT_SYM_DEF_OBJECT](#) in their `_PARMS` structure for specifying the cryptographic parameters for protecting child objects.

Table 233: Definition of `TPMU_PUBLIC_PARMS` Union

| 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) |
| mldsaDetail | TPMS_MLDSA_PARMS | TPM_ALG_MLDSA | sign |
| hash_mldsaDetail | TPMS_HASH_MLDSA_PARMS | TPM_ALG_HASH_MLDSA | sign |
| mlkemDetail | TPMS_MLKEM_PARMS | TPM_ALG_MLKEM | decrypt |

Note:
 [1] The 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.
 Support for both *sign* and *decrypt* on the same key was deprecated in version 185. See Part 0.

12.2.3.10 TPMT_PUBLIC_PARMS

This Table 234 structure is used in `TPM2_TestParms()` to validate that a set of algorithm parameters is supported by the TPM.

Table 234: 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 |

12.2.4 TPMT_PUBLIC

Table 235 defines the public area structure. The Name of the object is *nameAlg* concatenated with the digest of this structure using *nameAlg*.

Table 235: Definition of `TPMT_PUBLIC` Structure

| Parameter | Type | Description |
|-----------|---------------------------------|---|
| type | TPMI_ALG_PUBLIC | “algorithm” associated with this object |

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|------------------|-------------------|---|
| 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 <i>nameAlg</i> may be TPM_ALG_NULL. |
| objectAttributes | TPMA_OBJECT | attributes that, along with <i>type</i> , determine the manipulations of this object |
| authPolicy | TPM2B_DIGEST | optional policy for using this key The policy is computed using the <i>nameAlg</i> of the object. Note: This is 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. |

12.2.5 TPM2B_PUBLIC

This Table 236 sized buffer is used to embed a TPMT_PUBLIC in a load command and in any response that returns a public area.

Table 236: 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 <i>size</i> . 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. |

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|------------|--------------|---|
| publicArea | +TPMT_PUBLIC | the public area Note: The “+” indicates that the caller can specify that use of TPM_ALG_NULL is allowed for <i>nameAlg</i> . |

12.2.6 TPM2B_TEMPLATE

This Table 237 sized buffer is used to embed a TPMT_PUBLIC 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 Clause 12.2.3.2.

Table 237: Definition of TPM2B_TEMPLATE Structure

| Parameter | Type | Description |
|------------------------------------|--------|--------------------|
| size | UINT16 | size of publicArea |
| buffer[size]{:sizeof(TPMT_PUBLIC)} | BYTE | the public area |

12.3 Private Area Structures

12.3.1 Introduction

The structures in Clause 12.3 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.

12.3.2 Sensitive Data Structures

12.3.2.1 Introduction

The structures in Clause 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 Clause 12.3.2 shall be used.

12.3.2.2 TPM2B_PRIVATE_VENDOR_SPECIFIC

This Table 238 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 238: Definition of TPM2B_PRIVATE_VENDOR_SPECIFIC Structure

| Parameter | Type | Description |
|--|--------|-------------|
| size | UINT16 | |
| buffer[size]{:PRIVATE_VENDOR_SPECIFIC_BYTES} | BYTE | |

12.3.2.3 TPMU_SENSITIVE_COMPOSITE

Table 239 defines the TPMU_SENSITIVE_COMPOSITE union.

Table 239: Definition of TPMU_SENSITIVE_COMPOSITE Union

| 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 |
| mldsa | TPM2B_PRIVATE_KEY_MLDSA | TPM_ALG_MLDSA or TPM_ALG_HASH_MLDSA | the private seed ξ |
| mlkem | TPM2B_PRIVATE_KEY_MLKEM | TPM_ALG_MLKEM | the private seed $(d z)$ |
| any | TPM2B_PRIVATE_VENDOR_SPECIFIC | | vendor-specific size for key storage |

12.3.2.4 TPMT_SENSITIVE

Table 240 defines the TPMT_SENSITIVE Structure.

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 240: Definition of TPMT_SENSITIVE Structure

| Parameter | Type | Description |
|---------------|-----------------|---|
| sensitiveType | TPMI_ALG_PUBLIC | identifier for the sensitive area This shall be the same as the <i>type</i> 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 |

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|--------------------------|--------------------------|--------------------------------|
| [sensitiveType]sensitive | TPMU_SENSITIVE_COMPOSITE | the type-specific private data |

12.3.3 TPM2B_SENSITIVE

The Table 241 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:

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:

The unmarshaling function validates that *size* equals the size of the value that is unmarshaled.

Table 241: Definition of TPM2B_SENSITIVE Structure

| Parameter | Type | Description |
|---------------|----------------|--------------------------------------|
| size | UINT16 | size of the <i>private</i> structure |
| sensitiveArea | TPMT_SENSITIVE | an unencrypted sensitive area |

12.3.4 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” sub-clause “Symmetric Encryption.” in TPM 2.0 Part 1.

12.3.5 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” sub-clause “Integrity” in TPM 2.0 Part 1.

12.3.6 _PRIVATE

This Table 242 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 242: 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 |

12.3.7 TPM2B_PRIVATE

The Table 243 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 243: Definition of TPM2B_PRIVATE Structure

| Parameter | Type | Description |
|------------------------------------|--------|--------------------------------------|
| size | UINT16 | size of the <i>private</i> structure |
| buffer[size] { :sizeof(_PRIVATE) } | BYTE | an encrypted private area |

12.4 Identity Object

12.4.1 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.

12.4.2 TPMS_ID_OBJECT

This Table 244 structure is used for sizing the TPM2B_ID_OBJECT.

Table 244: Definition of TPMS_ID_OBJECT Structure

| Parameter | Type | Description |
|---------------|--------------|---|
| integrityHMAC | TPM2B_DIGEST | HMAC using the nameAlg of the storage key on the target TPM |

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|--------------|--------------|--|
| enclidentity | TPM2B_DIGEST | credential protector information returned if name matches the referenced object All of the <i>enclidentity</i> 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 <i>nameAlg</i> . However, if the size is larger, the ID object may not be usable on a TPM that has no digest larger than produced by <i>nameAlg</i> . |

12.4.3 TPM2B_ID_OBJECT

This Table 245 structure is an output from TPM2_MakeCredential() and is an input to TPM2_ActivateCredential().

Table 245: Definition of TPM2B_ID_OBJECT Structure

| Parameter | Type | Description |
|---|--------|---|
| size | UINT16 | size of the <i>credential</i> structure |
| credential[size]{:sizeof(TPMS_ID_OBJECT)} | BYTE | an encrypted credential area |

13 NV Storage Structures

13.1 TPM_NV_INDEX

A Table 246 TPM_NV_INDEX is used to reference a defined location in NV memory. Handles in this range use the digest of the public area of the Index as the Name of the entity in authorization computations.

A valid Index handle will have an MSO corresponding to one of the NV index handle types in Clause 7.2, i.e., one of TPM_HT_NV_INDEX, TPM_HT_EXTERNAL_NV, or TPM_HT_PERMANENT_NV.

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 246: Definition of (UINT32) TPM_NV_INDEX Bits

| Bit | Name | Definition |
|-------|--------|--|
| 23:00 | index | the Index of the NV location |
| 31:24 | TPM_HT | constant value indicating the NV Index handle type |

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.

13.2 TPM_NT

Table 247 lists the values of the TPM_NT field of a TPMA_NV. See Table 249 for usage.

Table 247: 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 <i>pinCount</i> that increments on a PIN authorization failure and a <i>pinLimit</i> |

(continued on next page)

(continued from previous page)

| Name | Value | Description |
|-----------------|-------|--|
| TPM_NT_PIN_PASS | 0x9 | PIN Pass - contains <i>pinCount</i> that increments on a PIN authorization success and a <i>pinLimit</i> |

All other TPM_NT values are reserved and TPM2_NV_DefineSpace() returns TPM_RC_ATTRIBUTES.

Note:

These values are compatible with previous versions of this specification, which used a bit map for this field.

Note:

This field described by Table 247 is 4 bits.

13.3 TPMS_NV_PIN_COUNTER_PARAMETERS

This Table 248 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 248: 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 <i>pinCount</i> at which the authValue authorization of the host TPM_NT_PIN_PASS or TPM_NT_PIN_FAIL index is locked out. |

13.4 TPMA_NV

This Table 249 NV Index attributes structure allows the TPM to keep track of the data and permissions to manipulate an NV Index. See Table 250 for a 64-bit version of this structure.

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 249: Definition of (UINT32) TPMA_NV Bits

| Bit | Name | Description |
|-----|--------------------|--|
| 0 | TPMA_NV_PPWRITE | SET (1): The Index data can be written or write-locked if Platform Authorization is provided. CLEAR (0): Writing or write-locking the Index data cannot be authorized with Platform Authorization. |
| 1 | TPMA_NV_OWNERWRITE | SET (1): The Index data can be written or write-locked if Owner Authorization is provided. CLEAR (0): Writing or write-locking the Index data cannot be authorized with Owner Authorization. |
| 2 | TPMA_NV_AUTHWRITE | SET (1): The Index data may be written or write-locked if the Index <i>authValue</i> is provided. CLEAR (0): Writing or write-locking the Index data cannot be authorized with the Index <i>authValue</i> . |

(continued on next page)

(continued from previous page)

| Bit | Name | Description |
|-----|-----------------------|--|
| 3 | TPMA_NV_POLICYWRITE | <p>SET (1): Authorizations to change the Index contents or write-lock that require USER role may be provided with a policy session.</p> <p>CLEAR (0): Authorizations to change the Index contents or write-lock that require USER role may not be provided with a policy session.</p> <p>Note:</p> <p>TPM2_NV_ChangeAuth() always requires that authorization be provided in a policy session.</p> |
| 7:4 | TPM_NT | <p>the type of the index.</p> <p>Note:</p> <p>A TPM is not required to support all TPM_NT values</p> |
| 9:8 | Reserved | <p>shall be zero</p> <p>reserved for future use</p> |
| 10 | TPMA_NV_POLICY_DELETE | <p>SET (1): Index may not be deleted unless the <i>authPolicy</i> is satisfied using TPM2_NV_UndefineSpaceSpecial().</p> <p>CLEAR (0): Index may be deleted with proper platform or owner authorization using TPM2_NV_UndefineSpace().</p> <p>Note:</p> <p>An Index with this attribute and a policy that cannot be satisfied (e.g., an Empty Policy or a policy that does not include the command code TPM_CC_NV_UndefineSpaceSpecial()) cannot be deleted. Application development using a software TPM is advised.</p> |
| 11 | TPMA_NV_WRITELOCKED | <p>SET (1): Index cannot be written.</p> <p>CLEAR (0): Index can be written.</p> |

(continued on next page)

(continued from previous page)

| Bit | Name | Description |
|-------|-----------------------|--|
| 12 | TPMA_NV_WRITEALL | <p>SET (1): A partial write of the Index data is not allowed. The write size shall match the defined space size.</p> <p>CLEAR (0): Partial writes are allowed. This setting is required if the <i>.dataSize</i> of the Index is larger than NV_MAX_BUFFER_SIZE for the implementation.</p> <p>Note: This attribute is only checked for TPM2_NV_Write().</p> |
| 13 | TPMA_NV_WRITEDEFINE | <p>SET (1): TPM2_NV_WriteLock() may be used to prevent further writes to this location.</p> <p>CLEAR (0): TPM2_NV_WriteLock() does not block subsequent writes if TPMA_NV_WRITE_STCLEAR is also CLEAR.</p> |
| 14 | TPMA_NV_WRITE_STCLEAR | <p>SET (1): TPM2_NV_WriteLock() may be used to prevent further writes to this location until the next TPM Reset or TPM Restart.</p> <p>CLEAR (0): TPM2_NV_WriteLock() does not block subsequent writes if TPMA_NV_WRITEDEFINE is also CLEAR.</p> |
| 15 | TPMA_NV_GLOBALLOCK | <p>SET (1): If TPM2_NV_GlobalWriteLock() is successful, TPMA_NV_WRITELOCKED is set.</p> <p>CLEAR (0): TPM2_NV_GlobalWriteLock() has no effect on the writing of the data at this Index.</p> |
| 16 | TPMA_NV_PPREAD | <p>SET (1): The Index data can be read or read-locked if Platform Authorization is provided.</p> <p>CLEAR (0): Reading or read-locking the Index data cannot be authorized with Platform Authorization.</p> |
| 17 | TPMA_NV_OWNERREAD | <p>SET (1): The Index data can be read or read-locked if Owner Authorization is provided.</p> <p>CLEAR (0): Reading or read-locking the Index data cannot be authorized with Owner Authorization.</p> |
| 18 | TPMA_NV_AUTHREAD | <p>SET (1): The Index data may be read or read-locked if the Index <i>authValue</i> is provided.</p> <p>CLEAR (0): Reading or read-locking the Index data cannot be authorized with the Index <i>authValue</i>.</p> |
| 19 | TPMA_NV_POLICYREAD | <p>SET (1): The Index data may be read or read-locked if the <i>authPolicy</i> is satisfied.</p> <p>CLEAR (0): Reading or read-locking the Index data cannot be authorized with the Index <i>authPolicy</i>.</p> |
| 24:20 | Reserved | shall be zero reserved for future use |

(continued on next page)

(continued from previous page)

| Bit | Name | Description |
|-----|------------------------|---|
| 25 | TPMA_NV_NO_DA | <p>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.</p> <p>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.</p> |
| 26 | TPMA_NV_ORDERLY | <p>SET (1): NV Index state is only saved to NV when the TPM performs an orderly shutdown (TPM2_Shutdown()). See also MAX_ORDERLY_COUNT.</p> <p>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).</p> |
| 27 | TPMA_NV_CLEAR_STCLEAR | <p>SET (1): TPMA_NV_WRITTEN for the Index is CLEAR by TPM Reset or TPM Restart.</p> <p>CLEAR (0): TPMA_NV_WRITTEN is not changed by TPM Restart.</p> <p>Note: This attribute can only be SET if TPM_NT is not TPM_NT_COUNTER.</p> |
| 28 | TPMA_NV_READLOCKED | <p>SET (1): Reads of the Index are blocked until the next TPM Reset or TPM Restart.</p> <p>CLEAR (0): Reads of the Index are allowed if proper authorization is provided.</p> |
| 29 | TPMA_NV_WRITTEN | <p>SET (1): Index has been written.</p> <p>CLEAR (0): Index has not been written.</p> |
| 30 | TPMA_NV_PLATFORMCREATE | <p>SET (1): This Index may be undefined with Platform Authorization but not with Owner Authorization.</p> <p>CLEAR (0): This Index may be undefined using Owner Authorization but not with Platform Authorization.</p> <p>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.</p> |
| 31 | TPMA_NV_READ_STCLEAR | <p>SET (1): TPM2_NV_ReadLock() may be used to SET TPMA_NV_READLOCKED for this Index.</p> <p>CLEAR (0): TPM2_NV_ReadLock() has no effect on this Index.</p> |

13.5 TPMA_NV_EXP

This Table 250 expanded NV Index attributes structure describes the expanded attributes that apply to certain types of NV indices. See Table 249 for the low 32 bits.

Note:

This structure was added in version 1.83.

Table 250: Definition of (UINT64) TPMA_NV_EXP Bits

| Bit | Name | Description |
|-------|-------------------------------|--|
| 31:0 | – | as defined in TPMA_NV |
| 32 | TPMA_EXTERNAL_NV_ENCRYPTION | SET (1): External NV index contents are encrypted. CLEAR (0): External NV index contents are not encrypted. |
| 33 | TPMA_EXTERNAL_NV_INTEGRITY | SET (1): External NV index contents are integrity-protected. CLEAR (0): External NV index contents are not integrity-protected. |
| 34 | TPMA_EXTERNAL_NV_ANTIROLLBACK | SET (1): External NV index contents are rollback-protected. CLEAR (0): External NV index contents are not rollback-protected. |
| 63:35 | Reserved | shall be zero reserved for future use |

13.6 TPMS_NV_PUBLIC

This Table 251 legacy structure describes an NV Index.

Note:

This is the legacy form of this structure, which only supports regular NV indices (TPM_HT_NV_INDEX).

The generalized form is [TPMT_NV_PUBLIC_2](#).

Table 251: Definition of TPMS_NV_PUBLIC Structure

| Name | Type | Description |
|------------|---|---|
| nvIndex | TPMI_RH_NV_LEGACY_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 <i>authPolicy</i> For an extend index, the hash algorithm used for the extend. |
| attributes | TPMA_NV | the Index attributes |

(continued on next page)

(continued from previous page)

| Name | Type | Description |
|-----------------------------|------------------------------|--|
| authPolicy | TPM2B_DIGEST | optional access policy for the Index The policy is computed using the <i>nameAlg</i> . Note: This is 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 |

13.7 TPM2B_NV_PUBLIC

This Table [252](#) legacy structure is used when a TPMS_NV_PUBLIC is sent on the TPM interface.

Note:

This is the legacy form of this structure, which only supports regular NV indices (TPM_HT_NV_INDEX).
The generalized form is [TPM2B_NV_PUBLIC_2](#).

Table 252: Definition of TPM2B_NV_PUBLIC Structure

| Name | Type | Description |
|----------|--------------------------------|-------------------------|
| size= | UINT16 | size of <i>nvPublic</i> |
| nvPublic | TPMS_NV_PUBLIC | the public area |

13.8 TPMS_NV_PUBLIC_EXP_ATTR

This Table [253](#) structure describes an NV Index with expanded NV attributes. This type is used by NV Index types that use those attributes.

Note:

This structure was added in version 1.83.

Table 253: Definition of TPMS_NV_PUBLIC_EXP_ATTR Structure

| Name | Type | Description |
|------------------------------|----------------------|--|
| nvIndex | TPMI_RH_NV_EXP_INDEX | the handle of the data area |
| nameAlg | TPMI_ALG_HASH | hash algorithm used to compute the Name of the Index This is used for the <i>authPolicy</i> . For an extend index, the hash algorithm used for the extend. |
| attributes | TPMA_NV_EXP | the Index attributes |
| authPolicy | TPM2B_DIGEST | optional access policy for the Index The policy is computed using the <i>nameAlg</i> Note: This is 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 |

13.9 TPMU_NV_PUBLIC_2

This Table 254 is the union of all values allowed in in the public field of a TPMT_NV_PUBLIC_2.

Note:

This structure was added in version 1.83.

Table 254: Definition of TPMU_NV_PUBLIC_2 Union

| Parameter | Type | Selector | Description |
|-------------|-------------------------|---------------------|-------------|
| nvIndex | TPMS_NV_PUBLIC | TPM_HT_NV_INDEX | |
| externalNV | TPMS_NV_PUBLIC_EXP_ATTR | TPM_HT_EXTERNAL_NV | |
| permanentNV | TPMS_NV_PUBLIC | TPM_HT_PERMANENT_NV | |

13.10 TPMT_NV_PUBLIC_2

This Table 255 data area contains an NV public area union, which supports any type of NV index handle.

Note:

This structure was added in version 1.83.

Note:

The *handleType* member is the same as the first byte of the public member, the *nvIndex* field.

Table 255: Definition of TPMT_NV_PUBLIC_2 Structure

| Parameter | Type | Description |
|------------------------|------------------|----------------------|
| handleType | TPM_HT | NV index handle type |
| [handleType]publicArea | TPMU_NV_PUBLIC_2 | NV index public area |

13.11 TPM2B_NV_PUBLIC_2

This Table 256 structure is used when a TPMT_NV_PUBLIC_2 is sent on the TPM interface.

Note:

This structure was added in version 1.83.

Table 256: Definition of TPM2B_NV_PUBLIC_2 Structure

| Name | Type | Description |
|----------|------------------|-------------------------|
| size= | UINT16 | size of <i>nvPublic</i> |
| nvPublic | TPMT_NV_PUBLIC_2 | the public area |

14 Context Data

14.1 Introduction

This clause 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.

14.2 TPM2B_CONTEXT_SENSITIVE

This Table 257 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 257: Definition of TPM2B_CONTEXT_SENSITIVE Structure

| Parameter | Type | Description |
|---------------------------------|--------|--------------------|
| size | UINT16 | |
| buffer[size]:{MAX_CONTEXT_SIZE} | BYTE | the sensitive data |

Note:

MAX_CONTEXT_SIZE (the maximum size of a TPM2B_CONTEXT_SENSITIVE) is the greater of PT_MAX_OBJECT_CONTEXT and PT_MAX_SESSION_CONTEXT, which can be queried via TPM2_GetCapability().

14.3 TPMS_CONTEXT_DATA

This Table 258 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 258: Definition of TPMS_CONTEXT_DATA Structure

| Parameter | Type | Description |
|-----------|-------------------------|---------------------|
| integrity | TPM2B_DIGEST | the integrity value |
| encrypted | TPM2B_CONTEXT_SENSITIVE | the sensitive area |

14.4 TPM2B_CONTEXT_DATA

This Table 259 structure is used in a TPMS_CONTEXT.

Table 259: Definition of TPM2B_CONTEXT_DATA Structure

| Parameter | Type | Description |
|--|--------|-------------|
| size | UINT16 | |
| buffer[size] {:sizeof(TPMS_CONTEXT_DATA)} | BYTE | |

14.5 TPMS_CONTEXT

This Table 260 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:

The Reference Code 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.

Note:

A use case for *stClear* is the following. At a TPM Restart, PCRs are reset to their default value. If an ancestor storage key requires PCR authorization (TPM2_PolicyPCR()), a child object is not loadable unless the ancestors back to the primary storage key have all been authorized using the approved PCR values. However, a saved context can be loaded independent of its ancestors. If the use case wants to block a context load, it sets the *stClear* attribute. This forces a load of all the ancestors and thus PCR verification.

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:

The Reference Code 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:

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 260: 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 58) |
| hierarchy | TPMI_RH_HIERARCHY+ | the hierarchy of the context |
| contextBlob | TPM2B_CONTEXT_DATA | the context data and integrity HMAC |

14.6 Parameters of TPMS_CONTEXT**14.6.1 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.

14.6.2 savedHandle

Table 58 describes the allowed TPMI_DH_SAVED values.

For a session, *savedHandle* is the handle that was assigned to the session when it was created.

For a transient object, rather than being the handle that was assigned to the object, *savedHandle* will be one of the three fixed handle values in Table 58:

- 0x80000000: the object was an ordinary transient object

- 0x80000001: the object was a sequence object
- 0x80000002: the object was a transient object with *stClear* SET

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.

14.6.3 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.

14.7 Context Protection

14.7.1 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.

14.7.2 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.

15 Creation Data

15.1 TPMS_CREATION_DATA

This Table 261 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 261 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 261: Definition of TPMS_CREATION_DATA Structure

| Parameter | Type | Description |
|---------------------|--------------------|--|
| pcrSelect | TPML_PCR_SELECTION | list indicating the PCR included in <i>pcrDigest</i> |
| pcrDigest | TPM2B_DIGEST | digest of the selected PCR using <i>nameAlg</i> of the object for which this structure is being created <i>pcrDigest.size</i> shall be zero if the <i>pcrSelect</i> list is empty. |
| locality | TPMA_LOCALITY | the locality at which the object was created |
| parentNameAlg | TPM_ALG_ID | <i>nameAlg</i> of the parent |
| parentName | TPM2B_NAME | Name of the parent at time of creation The size will match digest size associated with <i>parentNameAlg</i> unless it is TPM_ALG_NULL, in which case the size will be 4 and <i>parentName</i> will be the hierarchy handle. |
| parentQualifiedName | TPM2B_NAME | Qualified Name of the parent at the time of creation Size is the same as <i>parentName</i> . |
| outsideInfo | TPM2B_DATA | association with additional information added by the key creator This will be the contents of the <i>outsideInfo</i> parameter in TPM2_Create() or TPM2_CreatePrimary(). |

15.2 TPM2B_CREATION_DATA

This Table 262 structure is created by TPM2_Create() and TPM2_CreatePrimary(). It is never entered into the TPM and never has a size of zero.

Table 262: Definition of TPM2B_CREATION_DATA Structure

| Parameter | Type | Description |
|-----------|--------|---------------------------|
| size= | UINT16 | size of the creation data |

(continued on next page)

(continued from previous page)

| Parameter | Type | Description |
|------------------|--------------------|--------------------|
| creationData | TPMS_CREATION_DATA | |

16 Attached Component Structures

16.1 TPM_AT

These Table 263 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 263: 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 <i>outputData</i> 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 |

16.2 TPM_AE

These Table 265 constants are the TCG-defined error values returned by an AC.

Table 264: Definition of (UINT32) TPM_AE Constants

| Name | Value | Comments |
|-------------|------------|--|
| TPM_AE_NONE | 0x00000000 | in a command, a non-specific request for AC information; in a response, indicates that <i>outputData</i> is not meaningful |

16.3 TPMS_AC_OUTPUT

Table 265 TPMS_AC_OUTPUT is used to return information about an AC. The *tag* structure parameter indicates the type of the *data* value.

Table 265: Definition of TPMS_AC_OUTPUT Structure

| Parameter | Type | Description |
|-----------|--------|--|
| tag | TPM_AT | tag indicating the contents of <i>data</i> |
| data | UINT32 | the data returned from the AC |

16.4 TPML_AC_CAPABILITIES

This Table 266 list is only used in `TPM2_AC_GetCapability()`.

The values in the list are returned in TPM_AT order (see Table 263) with vendor-specific values returned after TCG defined values.

Note:

MAX_AC_CAPABILITIES = MAX_CAP_DATA / sizeof(TPMS_AC_OUTPUT)

Table 266: Definition of TPML_AC_CAPABILITIES Structure

| Parameter | Type | Description |
|---|----------------|--|
| count | UINT32 | number of values in the <i>acCapabilities</i> list; may be 0 |
| acCapabilities[count] {:MAX_AC_CAPABILITIES} | TPMS_AC_OUTPUT | a list of AC values |

References

- [1] "TCG Algorithm Registry." Trusted Computing Group, Feb. 2025. Available: <https://trustedcomputinggroup.org/resource/tcg-algorithm-registry/>
- [2] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch, "RFC 8017 - PKCS #1: RSA Cryptography Specifications Version 2.2." RFC Editor, Nov. 2016. Available: <https://www.rfc-editor.org/info/rfc8017>
- [3] "Information technology - Security techniques - Cryptographic techniques based on elliptic curves." ISO/IEC, Jul. 2016. Available: <https://www.iso.org/standard/65480.html>
- [4] "Module-Lattice-Based Key-Encapsulation Mechanism Standard." National Institute of Standards and Technology, Aug. 2024. Available: <https://doi.org/10.6028/NIST.FIPS.203>
- [5] "Module-Lattice-Based Digital Signature Standard." National Institute of Standards and Technology, Aug. 2024. Available: <https://doi.org/10.6028/NIST.FIPS.204>
- [6] "IT Security techniques - Hash-functions - Part 3: Dedicated hash-functions." ISO/IEC, Oct. 2018. Available: <https://www.iso.org/standard/67116.html>
- [7] "Information technology - Security techniques - Encryption algorithms - Part 3: Block ciphers." ISO/IEC, Dec. 2010. Available: <https://www.iso.org/standard/54531.html>
- [8] "SM4 Block Cipher Algorithm." Standardization Administration of the PRC, 2016.
- [9] "Information technology - Security techniques - Modes of operation for an n-bit block cipher." ISO/IEC, Jul. 2017. Available: <https://www.iso.org/standard/64575.html>
- [10] "Information security - Message authentication codes (MACs) - Part 2: Mechanisms using a dedicated hash-function." ISO/IEC, Jun. 2021. Available: <https://www.iso.org/standard/75296.html>
- [11] H. Krawczyk and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)." RFC Editor, May 2010. Available: <https://www.rfc-editor.org/info/rfc5869>
- [12] E. Barker, L. Chen, A. Roginsky, A. Vassilev, and R. Davis, "SP 800-56a: Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography." National Institute of Standards and Technology, Apr. 2018. Available: <https://doi.org/10.6028/NIST.SP.800-56Ar3>
- [13] E. Barker, L. Chen, and R. Davis, "SP 800-56c: Recommendation for Key-Derivation Methods in Key-Establishment Schemes." National Institute of Standards and Technology, Aug. 2020. Available: <https://doi.org/10.6028/NIST.SP.800-56Cr2>
- [14] L. Chen, "SP 800-108: Recommendation for Key Derivation Using Pseudorandom Functions." National Institute of Standards and Technology, Aug. 2022. Available: <http://csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf>
- [15] "IEEE Standard Specifications for Public-Key Cryptography - Amendment 1: Additional Techniques." IEEE, Sep. 2004. Available: <https://standards.ieee.org/ieee/1363a/2050/>
- [16] "IT Security techniques - Digital signatures with appendix - Part 3: Discrete logarithm based mechanisms." ISO/IEC, Nov. 2018. Available: <https://www.iso.org/standard/76382.html>
- [17] A. Langley, M. Hamburg, and S. Turner, "RFC 7748: Elliptic Curves for Security." RFC Editor, Jan. 2016. Available: <https://www.rfc-editor.org/info/rfc7748>
- [18] "Information Security Technology - Public Key Cryptographic Algorithm SM2 Based On Elliptic Curves - Part 1: General." Standardization Administration of the PRC, 2016. Available: <https://webstore.ansi.org/standards/spc/gb329182016>
- [19] "Information Security Technology - Public Key Cryptographic Algorithm SM2 Based On Elliptic Curves - Part 2: Digital Signature Algorithm." Standardization Administration of the PRC, 2016. Available: <https://webstore.ansi.org/standards/spc/gb329182016-1676169>
- [20] "Information Security Technology - Public Key Cryptographic Algorithm SM2 Based On Elliptic Curves - Part 3: Key exchange protocol." Standardization Administration of the PRC, 2016. Available: <https://webstore.ansi.org/standards/spc/gb329182016-1676171>
- [21] "Information Security Technology - Public Key Cryptographic Algorithm SM2 Based On Elliptic Curves - Part 4: Public key encryption algorithm." Standardization Administration of the PRC, 2016. Available: <https://webstore.ansi.org/standards/spc/gb329182016-1676170>

- [22] “Information Security Technology - Public Key Cryptographic Algorithm SM2 Based On Elliptic Curves - Part 5: Parameter definition.” Standardization Administration of the PRC, 2017. Available: <https://webstore.ansi.org/standards/spc/gb329182017>
- [23] S. Josefsson and I. Liusvaara, “RFC 8032: Edwards-Curve Digital Signature Algorithm (EdDSA).” RFC Editor, Jan. 2017. Available: <https://www.rfc-editor.org/info/rfc8032>
- [24] “IEEE Standard Specifications for Public-Key Cryptography.” IEEE, Aug. 2000. Available: <https://standards.ieee.org/ieee/1363/2049/>
- [25] R. Barnes, K. Bhargavan, B. Lipp, and C. Wood, “RFC 9180: Hybrid Public Key Encryption.” RFC Editor, Feb. 2022. Available: <https://www.rfc-editor.org/info/rfc9180>