INTERNATIONAL STANDARD

ISO/IEC 11889-2

Second edition
2015-12-15

Information technology — Trusted Platform Module Library —

Part 2: Structures

Technologies de l'information — Bibliothèque de module de plate-forme de confiance — Partie 2: Structures

Reference number
ISO/IEC 11889-2:2015(E)

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

CONTENTS

Foreword xv
Introduction xvi
1 Scope 1
2 Normative references 1
3 Terms and definitions 1
4 Symbols and abbreviated terms 1
5 Notation 1
5.1 Introduction 1
5.2 Named Constants 2
5.3 Data Type Aliases (typedefs) 3
5.4 Enumerations 3
5.5 Interface Type 4
5.6 Arrays 5
5.7 Structure Definitions 6
5.8 Conditional Types 7
5.9 Unions 8
5.9.1 Introduction 8
5.9.2 Union Definition 8
5.9.3 Union Instance 9
5.9.4 Union Selector Definition 10
5.10 Bit Field Definitions 11
5.11 Parameter Limits 12
5.12 Enumeration Macro 13
5.13 Size Checking 13
5.14 Data Direction 14
5.15 Structure Validations 15
5.16 Name Prefix Convention 15
5.17 Data Alignment 16
5.18 Parameter Unmarshaling Errors 16
6 Base Types 18
6.1 Primitive Types 18
6.2 Miscellaneous Types 18
7 Constants 19
7.1 TPM_SPEC (Specification Version Values) 19
7.2 TPM_GENERATED 19
7.3 TPM_ALG_ID 20
7.4 TPM_ECC_CURVE 24
7.5 TPM_CC (Command Codes) 24

7.5.1 Format 24
7.5.2 Description 25
7.5.3 TPM_CC Listing 26
7.6 TPM_RC (Response Codes) 29
7.6.1 Description 29
7.6.2 Response Code Formats 30
7.6.3 TPM_RC Values 33
7.7 TPM_CLOCK_ADJUST 38
7.8 TPM_EO (EA Arithmetic Operands) 38
7.9 TPM_ST (Structure Tags) 39
7.10 TPM_SU (Startup Type) 41
7.11 TPM_SE (Session Type) 41
7.12 TPM_CAP (Capabilities) 42
7.13 TPM_PT (Property Tag) 43
7.14 TPM_PT_PCR (PCR Property Tag) 48
7.15 TPM_PS (Platform Specific) 50
8 Handles 51
8.1 Introduction 51
8.2 TPM_HT (Handle Types) 51
8.3 Persistent Handle Sub-ranges 52
8.4 TPM_RH (Permanent Handles) 53
8.5 TPM_HC (Handle Value Constants) 54
9 Attribute Structures 56
9.1 Description 56
9.2 TPMA_ALGORITHM 56
9.3 TPMA_OBJECT (Object Attributes) 56
9.3.1 Introduction 56
9.3.2 Structure Definition 57
9.3.3 Attribute Descriptions 58
9.4 TPMA_SESSION (Session Attributes) 63
9.5 TPMA_LOCALITY (Locality Attribute) 64
9.6 TPMA_PERMANENT 65
9.7 TPMA_STARTUP_CLEAR 66
9.8 TPMA_MEMORY 67
9.9 TPMA_CC (Command Code Attributes) 68
9.9.1 Introduction 68
9.9.2 Structure Definition 68
9.9.3 Field Descriptions 68
10 Interface Types 71
10.1 Introduction 71
10.2 TPMI_YES_NO 71
10.3 TPMI_DH_OBJECT 71

10.4 TPMI_DH_PERSISTENT 72
10.5 TPMI_DH_ENTITY 72
10.6 TPMI_DH_PCR 73
10.7 TPMI_SH_AUTH_SESSION 73
10.8 TPMI_SH_HMAC 73
10.9 TPMI_SH_POLICY 73
10.10 TPMI_DH_CONTEXT 74
10.11 TPMI_RH_HIERARCHY 74
10.12 TPMI_RH_ENABLES 74
10.13 TPMI_RH_HIERARCHY_AUTH 75
10.14 TPMI_RH_PLATFORM 75
10.15 TPMI_RH_OWNER 75
10.16 TPMI_RH_ENDORSEMENT 76
10.17 TPMI_RH_PROVISION 76
10.18 TPMI_RH_CLEAR 76
10.19 TPMI_RH_NV_AUTH 77
10.20 TPMI_RH_LOCKOUT 77
10.21 TPMI_RH_NV_INDEX 77
10.22 TPMI_ALG_HASH 78
10.23 TPMI_ALG_ASYM (Asymmetric Algorithms) 78
10.24 TPMI_ALG_SYM (Symmetric Algorithms) 79
10.25 TPMI_ALG_SYM_OBJECT 79
10.26 TPMI_ALG_SYM_MODE 80
10.27 TPMI_ALG_KDF (Key and Mask Generation Functions) 80
10.28 TPMI_ALG_SIG_SCHEME 81
10.29 TPMI_ECC_KEY_EXCHANGE 81
10.30 TPMI_ST_COMMAND_TAG 81
11 Structure Definitions 83
11.1 TPMS_EMPTY 83
11.2 TPMS_ALGORITHM_DESCRIPTION 83
11.3 Hash/Digest Structures 84
11.3.1 TPMU_HA (Hash) 84
11.3.2 TPMT_HA 84
11.4 Sized Buffers 85
11.4.1 Introduction 85
11.4.2 TPM2B_DIGEST 85
11.4.3 TPM2B_DATA 86
11.4.4 TPM2B_NONCE 86
11.4.5 TPM2B_AUTH 86
11.4.6 TPM2B_OPERAND 86
11.4.7 TPM2B_EVENT 87
11.4.8 TPM2B_MAX_BUFFER 87
11.4.9 TPM2B_MAX_NV_BUFFER 87
11.4.10 TPM2B_TIMEOUT 88
11.4.11 TPM2B_IV 88
11.5 Names 88
11.5.1 Introduction 88
11.5.2 TPMU_NAME 88
11.5.3 TPM2B_NAME 89
11.6 PCR Structures 89
11.6.1 TPMS_PCR_SELECT 89

11.6.2 TPMS_PCR_SELECTION 90
11.7 Tickets 90
11.7.1 Introduction 90
11.7.2 A NULL Ticket 91
11.7.3 TPMT_TK_CREATION 92
11.7.4 TPMT_TK_VERIFIED 93
11.7.5 TPMT_TK_AUTH 94
11.7.6 TPMT_TK_HASHCHECK 95
11.8 Property Structures 95
11.8.1 TPMS_ALG_PROPERTY 95
11.8.2 TPMS_TAGGED_PROPERTY 95
11.8.3 TPMS_TAGGED_PCR_SELECT 96
11.9 Lists 96
11.9.1 TPML_CC 96
11.9.2 TPML_CCA 97
11.9.3 TPML_ALG 97
11.9.4 TPML_HANDLE 97
11.9.5 TPML_DIGEST 98
11.9.6 TPML_DIGEST_VALUES 98
11.9.7 TPM2B_DIGEST_VALUES 98
11.9.8 TPML_PCR_SELECTION 99
11.9.9 TPML_ALG_PROPERTY 99
11.9.10 TPML_TAGGED_TPM_PROPERTY 99
11.9.11 TPML_TAGGED_PCR_PROPERTY 100
11.9.12 TPML_ECC_CURVE 100
11.10 Capabilities Structures 100
11.10.1 TPMU_CAPABILITIES 100
11.10.2 TPMS_CAPABILITY_DATA 101
11.11 Clock/Counter Structures 101
11.11.1 PMS_CLOCK_INFO 101
11.11.2 *Clock* 101
11.11.3 *ResetCount* 101
11.11.4 *RestartCount* 102
11.11.5 *Safe* 102
11.11.6 TPMS_TIME_INFO 102
11.12 TPM Attestation Structures 103
11.12.1 Introduction 103
11.12.2 TPMS_TIME_ATTEST_INFO 103
11.12.3 TPMS_CERTIFY_INFO 103
11.12.1 TPMS_QUOTE_INFO 103
11.12.2 TPMS_COMMAND_AUDIT_INFO 104
11.12.3 TPMS_SESSION_AUDIT_INFO 104
11.12.4 TPMS_CREATION_INFO 104
11.12.5 TPMS_NV_CERTIFY_INFO 104
11.12.6 TPMI_ST_ATTEST 105
11.12.7 TPMU_ATTEST 105
11.12.8 TPMS_ATTEST 105
11.12.9 TPM2B_ATTEST 106
11.13 Authorization Structures 106
11.13.1 Introduction 106
11.13.2 TPMS_AUTH_COMMAND 106
11.13.3 TPMS_AUTH_RESPONSE 106
12 Algorithm Parameters and Structures 107

12.1 Symmetric 107
12.1.1 Introduction 107
12.1.2 TPMI_AES_KEY_BITS 107
12.1.3 TPMI_SM4_KEY_BITS 107
12.1.4 TPMI_CAMELLIA KEY_BITS 108
12.1.5 TPMU_SYM_KEY_BITS 108
12.1.6 TPMU_SYM_MODE 108
12.1.7 TPMU_SYM_DETAILS 109
12.1.8 TPMT_SYM_DEF 109
12.1.9 TPMT_SYM_DEF_OBJECT 110
12.1.10 TPM2B_SYM_KEY 110
12.1.11 TPMS_SYMCIPHER_PARMS 110
12.1.12 TPM2B_SENSITIVE_DATA 110
12.1.13 TPMS_SENSITIVE_CREATE 111
12.1.14 TPM2B_SENSITIVE_CREATE 111
12.1.15 TPMS_SCHEME_SIGHASH 112
12.1.16 TPMI_ALG_HASH_SCHEME 112
12.1.17 HMAC_SIG_SCHEME 112
12.1.18 TPMS_SCHEME_XOR 113
12.1.19 TPMU_SCHEME_HMAC 113
12.1.20 TPMT_KEYEDHASH_SCHEME 113
12.2 Asymmetric 114
12.2.1 Signing Schemes 114
12.2.2 Encryption Schemes 116
12.2.3 Key Derivation Schemes 116
12.2.4 RSA 119
12.2.5 ECC 122
12.3 Signatures 124
12.3.1 TPMS_SIGNATURE_RSASSA 124
12.3.2 TPMS_SIGNATURE_RSAPSS 124
12.3.3 TPMS_SIGNATURE_ECDSA 125
12.3.4 TPMU_SIGNATURE 125
12.3.5 TPMT_SIGNATURE 126
12.4 Key/Secret Exchange 126
12.4.1 Introduction 126
12.4.2 TPMU_ENCRYPTED_SECRET 126
12.4.3 TPM2B_ENCRYPTED_SECRET 127
13 Key/Object Complex 128
13.1 Introduction 128
13.2 Public Area Structures 128
13.2.1 Description 128
13.2.2 TPMI_ALG_PUBLIC 128
13.2.3 Type-Specific Parameters 128
13.2.4 TPMT_PUBLIC 132
13.2.5 TPM2B_PUBLIC 132
13.3 Private Area Structures 133
13.3.1 Introduction 133
13.3.2 Sensitive Data Structures 133
13.3.3 TPM2B_SENSITIVE 134
13.3.4 Encryption 135
13.3.5 Integrity 135
13.3.6 _PRIVATE 135
13.3.7 TPM2B_PRIVATE 135

13.4 Identity Object 136
13.4.1 Description 136
13.4.2 _ID_OBJECT 136
13.4.3 TPM2B_ID_OBJECT 136
14 NV Storage Structures 137
14.1 TPM_NV_INDEX 137
14.2 TPMA_NV (NV Index Attributes) 138
14.3 TPMS_NV_PUBLIC 141
14.4 TPM2B_NV_PUBLIC 141
15 Context Data 142
15.1 Introduction 142
15.2 TPM2B_CONTEXT_SENSITIVE 142
15.3 TPMS_CONTEXT_DATA 142
15.4 TPM2B_CONTEXT_DATA 142
15.5 TPMS_CONTEXT 143
15.6 Parameters of TPMS_CONTEXT 143
15.6.1 *sequence* 143
15.6.2 *savedHandle* 144
15.6.3 *hierarchy* 145
15.7 Context Protection 145
15.7.1 Context Integrity 145
15.7.2 Context Confidentiality 145
16 Creation Data 146
16.1 TPMS_CREATION_DATA 146
16.2 TPM2B_CREATION_DATA 146
Annex A (informative) Algorithm Constants 147
A.1 Introduction 147
A.2 Allowed Hash Algorithms 147
A.2.1 SHA1 147
A.2.2 SHA256 147
A.2.3 SHA384 147
A.2.4 SHA512 148
A.2.5 SM3_256 148
A.3 Architectural Limits 148
Annex B (informative) Implementation Definitions 149
B.1 Introduction 149
B.2 Logic Values 149
B.3 Processor Values 149
B.4 Implemented Algorithms 150
B.5 Implemented Commands 151
B.6 Algorithm Constants 154
B.6.1 RSA 154
B.6.2 ECC 154

B.6.3 AES 154
B.6.4 SM4 154
B.6.5 CAMELLIA 155
B.6.6 Symmetric 155
B.7 Implementation Specific Values 156
Bibliography 159

Tables

Table 1 — Name Prefix Convention 15
Table 2 — Unmarshaling Errors 17
Table 3 — Definition of Base Types 18
Table 4 — Definition of Types for Documentation Clarity 18
Table 5 — Definition of (UINT32) TPM_SPEC Constants <> 19
Table 6 — Definition of (UINT32) TPM_GENERATED Constants <O> 19
Table 7 — Legend for TPM_ALG_ID Table 20
Table 8 — Definition of (UINT16) TPM_ALG_ID Constants <IN/OUT, S> 21
Table 9 — Definition of (UINT16) {ECC} TPM_ECC_CURVE Constants <IN/OUT, S> 24
Table 10 — TPM Command Format Fields Description 24
Table 11 — Legend for Command Code Tables 25
Table 12 — Definition of (UINT32) TPM_CC Constants (Numeric Order) <IN/OUT, S> 26
Table 13 — Format-Zero Response Codes 31
Table 14 — Format-One Response Codes 32
Table 15 — Response Code Groupings 32
Table 16 — Definition of (UINT32) TPM_RC Constants (Actions) <OUT> 33
Table 17 — Definition of (INT8) TPM_CLOCK_ADJUST Constants <IN> 38
Table 18 — Definition of (UINT16) TPM_EO Constants <IN/OUT> 38
Table 19 — Definition of (UINT16) TPM_ST Constants <IN/OUT, S> 39
Table 20 — Definition of (UINT16) TPM_SU Constants <IN> 41
Table 21 — Definition of (UINT8) TPM_SE Constants <IN> 41
Table 22 — Definition of (UINT32) TPM_CAP Constants 42
Table 23 — Definition of (UINT32) TPM_PT Constants <IN/OUT, S> 43
Table 24 — Definition of (UINT32) TPM_PT_PCR Constants <IN/OUT, S> 48
Table 25 — Definition of (UINT32) TPM_PS Constants <OUT> 50
Table 26 — Definition of Types for Handles 51
Table 27 — Definition of (UINT8) TPM_HT Constants <S> 51
Table 28 — Definition of (TPM_HANDLE) TPM_RH Constants <S> 53
Table 29 — Definition of (TPM_HANDLE) TPM_HC Constants <S> 55
Table 30 — Definition of (UINT32) TPMA_ALGORITHM Bits 56
Table 31 — Definition of (UINT32) TPMA_OBJECT Bits 57
Table 32 — Definition of (UINT8) TPMA_SESSION Bits <IN/OUT> 63
Table 33 — Definition of (UINT8) TPMA_LOCALITY Bits <IN/OUT> 65
Table 34 — Definition of (UINT32) TPMA_PERMANENT Bits <OUT> 65
Table 35 — Definition of (UINT32) TPMA_STARTUP_CLEAR Bits <OUT> 66
Table 36 — Definition of (UINT32) TPMA_MEMORY Bits <Out> 67
Table 37 — Definition of (TPM_CC) TPMA_CC Bits <OUT> 68

Table 38 — Definition of (BYTE) TPMI_YES_NO Type 71
Table 39 — Definition of (TPM_HANDLE) TPMI_DH_OBJECT Type 71
Table 40 — Definition of (TPM_HANDLE) TPMI_DH_PERSISTENT Type 72
Table 41 — Definition of (TPM_HANDLE) TPMI_DH_ENTITY Type <IN> 72
Table 42 — Definition of (TPM_HANDLE) TPMI_DH_PCR Type <IN> 73
Table 43 — Definition of (TPM_HANDLE) TPMI_SH_AUTH_SESSION Type <IN/OUT> 73
Table 44 — Definition of (TPM_HANDLE) TPMI_SH_HMAC Type <IN/OUT> 73
Table 45 — Definition of (TPM_HANDLE) TPMI_SH_POLICY Type <IN/OUT> 73
Table 46 — Definition of (TPM_HANDLE) TPMI_DH_CONTEXT Type 74
Table 47 — Definition of (TPM_HANDLE) TPMI_RH_HIERARCHY Type 74
Table 48 — Definition of (TPM_HANDLE) TPMI_RH_ENABLES Type 74
Table 49 — Definition of (TPM_HANDLE) TPMI_RH_HIERARCHY_AUTH Type <IN> 75
Table 50 — Definition of (TPM_HANDLE) TPMI_RH_PLATFORM Type <IN> 75
Table 51 — Definition of (TPM_HANDLE) TPMI_RH_OWNER Type <IN> 75
Table 52 — Definition of (TPM_HANDLE) TPMI_RH_ENDORSEMENT Type <IN> 76
Table 53 — Definition of (TPM_HANDLE) TPMI_RH_PROVISION Type <IN> 76
Table 54 — Definition of (TPM_HANDLE) TPMI_RH_CLEAR Type <IN> 76
Table 55 — Definition of (TPM_HANDLE) TPMI_RH_NV_AUTH Type <IN> 77
Table 56 — Definition of (TPM_HANDLE) TPMI_RH_LOCKOUT Type <IN> 77
Table 57 — Definition of (TPM_HANDLE) TPMI_RH_NV_INDEX Type <IN/OUT> 77
Table 58 — Definition of (TPM_ALG_ID) TPMI_ALG_HASH Type 78
Table 59 — Definition of (TPM_ALG_ID) TPMI_ALG_ASYM Type 78
Table 60 — Definition of (TPM_ALG_ID) TPMI_ALG_SYM Type 79
Table 61 — Definition of (TPM_ALG_ID) TPMI_ALG_SYM_OBJECT Type 79
Table 62 — Definition of (TPM_ALG_ID) TPMI_ALG_SYM_MODE Type 80
Table 63 — Definition of (TPM_ALG_ID) TPMI_ALG_KDF Type 80
Table 64 — Definition of (TPM_ALG_ID) TPMI_ALG_SIG_SCHEME Type 81
Table 65 — Definition of (TPM_ALG_ID) TPMI_ECC_KEY_EXCHANGE Type 81
Table 66 — Definition of (TPM_ST) TPMI_ST_COMMAND_TAG Type 81
Table 67 — Definition of TPMS_EMPTY Structure <IN/OUT> 83
Table 68 — Definition of TPMS_ALGORITHM_DESCRIPTION Structure <OUT> 83
Table 69 — Definition of TPMU_HA Union <IN/OUT, S> 84
Table 70 — Definition of TPMT_HA Structure <IN/OUT> 84
Table 71 — Definition of TPM2B_DIGEST Structure 85
Table 72 — Definition of TPM2B_DATA Structure 86
Table 73 — Definition of Types for TPM2B_NONCE 86
Table 74 — Definition of Types for TPM2B_AUTH 86
Table 75 — Definition of Types for TPM2B_OPERAND 86
Table 76 — Definition of TPM2B_EVENT Structure 87

Table 77 — Definition of TPM2B_MAX_BUFFER Structure 87
Table 78 — Definition of TPM2B_MAX_NV_BUFFER Structure 87
Table 79 — Definition of TPM2B_TIMEOUT Structure <IN/OUT> 88
Table 80 — Definition of TPM2B_IV Structure <IN/OUT> 88
Table 81 — Definition of TPMU_NAME Union <> 88
Table 82 — Definition of TPM2B_NAME Structure 89
Table 83 — Definition of TPMS_PCR_SELECT Structure 90
Table 84 — Definition of TPMS_PCR_SELECTION Structure 90
Table 85 — Values for *proof* Used in Tickets 91
Table 86 — General Format of a Ticket 91
Table 87 — Definition of TPMT_TK_CREATION Structure 92
Table 88 — Definition of TPMT_TK_VERIFIED Structure 93
Table 89 — Definition of TPMT_TK_AUTH Structure 94
Table 90 — Definition of TPMT_TK_HASHCHECK Structure 95
Table 91 — Definition of TPMS_ALG_PROPERTY Structure <OUT> 95
Table 92 — Definition of TPMS_TAGGED_PROPERTY Structure <OUT> 95
Table 93 — Definition of TPMS_TAGGED_PCR_SELECT Structure <OUT> 96
Table 94 — Definition of TPML_CC Structure 96
Table 95 — Definition of TPML_CCA Structure <OUT> 97
Table 96 — Definition of TPML_ALG Structure 97
Table 97 — Definition of TPML_HANDLE Structure <OUT> 97
Table 98 — Definition of TPML_DIGEST Structure 98
Table 99 — Definition of TPML_DIGEST_VALUES Structure 98
Table 100 — Definition of TPM2B_DIGEST_VALUES Structure 98
Table 101 — Definition of TPML_PCR_SELECTION Structure 99
Table 102 — Definition of TPML_ALG_PROPERTY Structure <OUT> 99
Table 103 — Definition of TPML_TAGGED_TPM_PROPERTY Structure <OUT> 99
Table 104 — Definition of TPML_TAGGED_PCR_PROPERTY Structure <OUT> 100
Table 105 — Definition of {ECC} TPML_ECC_CURVE Structure <OUT> 100
Table 106 — Definition of TPMU_CAPABILITIES Union <OUT> 100
Table 107 — Definition of TPMS_CAPABILITY_DATA Structure <OUT> 101
Table 108 — Definition of TPMS_CLOCK_INFO Structure 101
Table 109 — Definition of TPMS_TIME_INFO Structure 102
Table 110 — Definition of TPMS_TIME_ATTEST_INFO Structure <OUT> 103
Table 111 — Definition of TPMS_CERTIFY_INFO Structure <OUT> 103
Table 112 — Definition of TPMS_QUOTE_INFO Structure <OUT> 103
Table 113 — Definition of TPMS_COMMAND_AUDIT_INFO Structure <OUT> 104
Table 114 — Definition of TPMS_SESSION_AUDIT_INFO Structure <OUT> 104
Table 115 — Definition of TPMS_CREATION_INFO Structure <OUT> 104

Table 116 — Definition of TPMS_NV_CERTIFY_INFO Structure <OUT> 104
Table 117 — Definition of (TPM_ST) TPMI_ST_ATTEST Type <OUT> 105
Table 118 — Definition of TPMU_ATTEST Union <OUT> 105
Table 119 — Definition of TPMS_ATTEST Structure <OUT> 105
Table 120 — Definition of TPM2B_ATTEST Structure <OUT> 106
Table 121 — Definition of TPMS_AUTH_COMMAND Structure <IN> 106
Table 122 — Definition of TPMS_AUTH_RESPONSE Structure <OUT> 106
Table 123 — Definition of {AES} (TPM_KEY_BITS) TPMI_AES_KEY_BITS Type 107
Table 124 — Definition of {SM4} (TPM_KEY_BITS) TPMI_SM4_KEY_BITS Type 107
Table 125 — Definition of {CAMELLIA} (TPM_KEY_BITS) TPMI_CAMELLIA_KEY_BITS Type 108
Table 126 — Definition of TPMU_SYM_KEY_BITS Union 108
Table 127 — Definition of TPMU_SYM_MODE Union 108
Table 128 —xDefinition of TPMU_SYM_DETAILS Union 109
Table 129 — Definition of TPMT_SYM_DEF Structure 109
Table 130 — Definition of TPMT_SYM_DEF_OBJECT Structure 110
Table 131 — Definition of TPM2B_SYM_KEY Structure 110
Table 132 — Definition of TPMS_SYMCIPHER_PARMS Structure 110
Table 133 — Definition of TPM2B_SENSITIVE_DATA Structure 111
Table 134 — Definition of TPMS_SENSITIVE_CREATE Structure <IN> 111
Table 135 — Definition of TPM2B_SENSITIVE_CREATE Structure <IN, S> 111
Table 136 — Definition of TPMS_SCHEME_SIGHASH Structure 112
Table 137 — Definition of (TPM_ALG_ID) TPMI_ALG_KEYEDHASH_SCHEME Type 112
Table 138 — Definition of Types for HMAC_SIG_SCHEME 112
Table 139 — Definition of TPMS_SCHEME_XOR Structure 113
Table 140 — Definition of TPMU_SCHEME_KEYEDHASH Union <IN/OUT, S> 113
Table 141 — Definition of TPMT_KEYEDHASH_SCHEME Structure 113
Table 142 — Definition of {RSA} Types for RSA_SIG_SCHEMES 114
Table 143 — Definition of {ECC} Types for ECC_SIG_SCHEMES 114
Table 144 — Definition of {ECC} TPMS_SCHEME_ECDAA Structure 114
Table 145 — Definition of TPMU_SIG_SCHEME Union <IN/OUT, S> 115
Table 146 — Definition of TPMT_SIG_SCHEME Structure 115
Table 147 — Definition of {RSA} TPMS_SCHEME_OAEP Structure 116
Table 148 — Definition of {ECC} TPMS_SCHEME_ECDH Structure 116
Table 149 — Definition of TPMS_SCHEME_MGF1 Structure 116
Table 150 — Definition of {ECC} TPMS_SCHEME_KDF1_SP800_56a Structure 116
Table 151 — Definition of TPMS_SCHEME_KDF2 Structure 117
Table 152 — Definition of TPMS_SCHEME_KDF1_SP800_108 Structure 117
Table 153 — Definition of TPMU_KDF_SCHEME Union <IN/OUT, S> 117
Table 154 — Definition of TPMT_KDF_SCHEME Structure 117

Table 155 — Definition of (TPM_ALG_ID) TPMI_ALG_ASYM_SCHEME Type <> ... 118
Table 156 — Definition of TPMU_ASYM_SCHEME Union ... 118
Table 157 — Definition of TPMT_ASYM_SCHEME Structure <> ... 119
Table 158 — Definition of (TPM_ALG_ID) {RSA} TPMI_ALG_RSA_SCHEME Type ... 119
Table 159 — Definition of {RSA} TPMT_RSA_SCHEME Structure ... 119
Table 160 — Definition of (TPM_ALG_ID) {RSA} TPMI_ALG_RSA_DECRYPT Type ... 120
Table 161 — Definition of {RSA} TPMT_RSA_DECRYPT Structure ... 120
Table 162 — Definition of {RSA} TPM2B_PUBLIC_KEY_RSA Structure ... 120
Table 163 — Definition of {RSA} (TPM_KEY_BITS) TPMI_RSA_KEY_BITS Type ... 121
Table 164 — Definition of {RSA} TPM2B_PRIVATE_KEY_RSA Structure ... 121
Table 165 — Definition of {ECC} TPM2B_ECC_PARAMETER Structure ... 122
Table 166 — Definition of {ECC} TPMS_ECC_POINT Structure ... 122
Table 167 — Definition of {ECC} TPM2B_ECC_POINT Structure ... 122
Table 168 — Definition of (TPM_ALG_ID) {ECC} TPMI_ALG_ECC_SCHEME Type ... 123
Table 169 — Definition of {ECC} (TPM_ECC_CURVE) TPMI_ECC_CURVE Type ... 123
Table 170 — Definition of (TPMT_SIG_SCHEME) {ECC} TPMT_ECC_SCHEME Structure ... 123
Table 171 — Definition of {ECC} TPMS_ALGORITHM_DETAIL_ECC Structure <OUT> ... 124
Table 172 — Definition of {RSA} TPMS_SIGNATURE_RSASSA Structure ... 124
Table 173 — Definition of {RSA} TPMS_SIGNATURE_RSAPSS Structure ... 125
Table 174 — Definition of {ECC} TPMS_SIGNATURE_ECDSA Structure ... 125
Table 175 — Definition of TPMU_SIGNATURE Union <IN/OUT, S> ... 125
Table 176 — Definition of TPMT_SIGNATURE Structure ... 126
Table 177 — Definition of TPMU_ENCRYPTED_SECRET Union <S> ... 126
Table 178 — Definition of TPM2B_ENCRYPTED_SECRET Structure ... 127
Table 179 — Definition of (TPM_ALG_ID) TPMI_ALG_PUBLIC Type ... 128
Table 180 — Definition of TPMU_PUBLIC_ID Union <IN/OUT, S> ... 129
Table 181 — Definition of TPMS_KEYEDHASH_PARMS Structure ... 129
Table 182 — Definition of TPMS_ASYM_PARMS Structure <> ... 130
Table 183 — Definition of {RSA} TPMS_RSA_PARMS Structure ... 130
Table 184 — Definition of {ECC} TPMS_ECC_PARMS Structure ... 131
Table 185 — Definition of TPMU_PUBLIC_PARMS Union <IN/OUT, S> ... 131
Table 186 — Definition of TPMT_PUBLIC_PARMS Structure ... 132
Table 187 — Definition of TPMT_PUBLIC Structure ... 132
Table 188 — Definition of TPM2B_PUBLIC Structure ... 132
Table 189 — Definition of TPM2B_PRIVATE_VENDOR_SPECIFIC Structure<> ... 133
Table 190 — Definition of TPMU_SENSITIVE_COMPOSITE Union <IN/OUT, S> ... 133
Table 191 — Definition of TPMT_SENSITIVE Structure ... 134
Table 192 — Definition of TPM2B_SENSITIVE Structure <IN/OUT> ... 134
Table 193 — Definition of _PRIVATE Structure <> ... 135

Table 194 — Definition of TPM2B_PRIVATE Structure <IN/OUT, S> 135
Table 195 — Definition of _ID_OBJECT Structure <> 136
Table 196 — Definition of TPM2B_ID_OBJECT Structure <IN/OUT> 136
Table 197 — Definition of (UINT32) TPM_NV_INDEX Bits <> 137
Table 198 — Definition of (UINT32) TPMA_NV Bits 139
Table 199 — Definition of TPMS_NV_PUBLIC Structure 141
Table 200 — Definition of TPM2B_NV_PUBLIC Structure 141
Table 201 — Definition of TPM2B_CONTEXT_SENSITIVE Structure <IN/OUT> 142
Table 202 — Definition of TPMS_CONTEXT_DATA Structure <IN/OUT, S> 142
Table 203 — Definition of TPM2B_CONTEXT_DATA Structure <IN/OUT> 142
Table 204 — Definition of TPMS_CONTEXT Structure 143
Table 205 — Context Handle Values 144
Table 206 — Definition of TPMS_CREATION_DATA Structure <OUT> 146
Table 207 — Definition of TPM2B_CREATION_DATA Structure <OUT> 146
Table A.1 — Defines for SHA1 Hash Values 147
Table A.2 — Defines for SHA256 Hash Values 147
Table A.3 — Defines for SHA384 Hash Values 147
Table A.4 — Defines for SHA512 Hash Values 148
Table A.5 — Defines for SM3_256 Hash Values 148
Table A.6 — Defines for Architectural Limits Values 148
Table B.1 — Defines for Logic Values 149
Table B.2 — Defines for Processor Values 149
Table B.3 — Defines for Implemented Algorithms 150
Table B.4 — Defines for Implemented Commands 151
Table B.5 — Defines for RSA Algorithm Constants 154
Table B.6 — Defines for ECC Algorithm Constants 154
Table B.7 — Defines for AES Algorithm Constants 154
Table B.8 — Defines for SM4 Algorithm Constants 154
Table B.9 — Defines for CAMELLIA Algorithm Constants 155
Table B.10 — Defines for Symmetric Algorithm Constants 155
Table B.11 — Defines for Implementation Values 156

Figures

Figure 1 — Command Format 24
Figure 2 — Format-Zero Response Codes 30
Figure 3 — Format-One Response Codes 31
Figure 4 — ISO/IEC 11889 (first edition) TPM_NV_INDEX 137
Figure 5 — ISO/IEC 11889 TPM_NV_INDEX 137

Foreword

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

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

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

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

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

ISO/IEC 11889-2 was prepared by the Trusted Computing Group (TCG) and was adopted, under the PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

This second edition cancels and replaces the first edition (ISO/IEC 11889-2:2009), which has been technically revised.

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

– *Part 1: Architecture*

– *Part 2: Structures*

– *Part 3: Commands*

– *Part 4: Supporting routines*

Introduction

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

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

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

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

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

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

Information technology — Trusted Platform Module Library — Part 2: Structures

1 Scope

This part of ISO/IEC 11889 contains the definitions of the constants, flags, structure, and union definitions used to communicate with the TPM. Values defined in this part of ISO/IEC 11889 are used by the TPM commands defined in ISO/IEC 11899-3 and by the functions in ISO/IEC 11889-4.

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 Normative references

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

- ISO/IEC 9797-2, *Information technology -- Security techniques -- Message Authentication Codes (MACs) -- Part 2: Mechanisms using a dedicated hash-function*
- ISO/IEC 10116:2006, *Information technology — Security techniques — Modes of operation for an n-bit block cipher*
- ISO/IEC 11889-1, *Information technology — Trusted Platform Module Library — Part 1: Architecture*
- ISO/IEC 11889-3, *Information technology — Trusted Platform Module Library — Part 3: Commands*
- ISO/IEC 11889-4, *Information technology — Trusted Platform Module Library — Part 4: Supporting routines*
- TCG Algorithm Registry, available at <http://www.trustedcomputinggroup.org/resources/tcg_algorithm_registry>

3 Terms and definitions

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

4 Symbols and abbreviated terms

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

5 Notation

5.1 Introduction

The information in this part of ISO/IEC 11889 is formatted so that it may be converted to standard computer-language formats by an automated process. The purpose of this automated process is to minimize the transcription errors that often occur during the conversion process.

For the purposes of this part of ISO/IEC 11889, the conventions given in ISO/IEC 11889-1 apply.

In addition, the conventions and notations in clause 5 describe the representation of various data so that it is both human readable and amenable to automated processing.

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

NOTE 1 In the examples in clause 5, the unmarshaling routines are shown as returning `bool`. In the code of the reference implementation, the return value is a TPM_RC. A `bool` is used in the examples, because the meaning of a TPM_RC is not yet defined.

NOTE 2 The unmarshaling code examples are the actual code that would be produced by the automatic code generator used in the construction of the reference code. The actual code contains additional parameter checking that is omitted for clarity of the principle being illustrated. Actual examples of the code are found in ISO/IEC 11889-4.

5.2 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 ISO/IEC 11889, 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. The title will include the data type of the constants in parentheses.

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

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

EXAMPLE 1

Table xx — Definition of (UINT16) COUNTING Constants

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

EXAMPLE 2

```
/* The C language equivalent of the constants from the table above */
typedef     UINT16      COUNTING;
#define     first       1
#define     second      0x0002
#define     third       3
#define     fourth      0x0004
```


5.3 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 ISO/IEC 11889, defined types have names that use all upper-case characters.

The table in Example 1 shows how typedefs would be defined in ISO/IEC 11889 and Example 2 shows the C-compatible code that might be produced from that table by an automated process.

EXAMPLE 1

Table xx — Definition of Types for Some Purpose

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

EXAMPLE 2

```
/* C language equivalent of the typedefs from the table above */
typedef unsigned short    UINT16;
typedef UINT16            SOME_TYPE;
typedef unsigned long     UINT32;
typedef UINT32            LAST_TYPE;
```

5.4 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 Example 1 shows how an enumeration would be defined in ISO/IEC 11889. Example 2 shows the C code that might be produced from that table by an automated process.

EXAMPLE 1

Table xx — Definition of (UINT16) CARD_SUIT Values

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

EXAMPLE 2

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

5.5 Interface Type

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

EXAMPLE 1

Table xx — Definition of (CARD_SUIT) RED_SUIT Type

Values	Comments
HEARTS	
DIAMONDS	
#TPM_RC_SUIT	response code returned when the unmarshaling of this type fails
NOTE	TPM_RC_SUIT is an example and no such response code is actually defined in ISO/IEC 11889.

EXAMPLE 2

```
/* Validation code that might be automatically generated from table above */
if((*target != HEARTS) && (*target != DIAMONDS))
    return TPM_RC_SUIT;
```

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

To illustrate, assume that the implementation only supports two sizes (1024 and 2048 bits) for keys associated with some algorithm (MY algorithm). In the implementation Annex B a named list would be created.

EXAMPLE 3

Table xx — Defines for MY Algorithm Constants

Name	Value	Comments
MY_KEY_SIZES_BITS	{1024, 2048}	braces because this is a list value

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

EXAMPLE 4

Table xx — Definition of (UINT16) MY_KEY_BITS Type

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

EXAMPLE 5

Table xx — Definition of (UINT16) MY_KEY_BITS Type

Parameter	Description
$MY_KEY_SIZES_BITS	the number of bits in the supported key

5.6 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 Example 1 shows how a structure containing fixed and variable-length arrays would be defined in ISO/IEC 11889. Example 2 shows the C code that might be produced from that table by an automated process.

EXAMPLE 1

Table xx — Definition of A_STRUCT Structure

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

EXAMPLE 2

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


5.7 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 Example 1 shows an example of how a structure would be defined in ISO/IEC 11889 and Example 2 shows the C code that might be produced from the table by an automated process. Example 3 illustrates the type of unmarshaling code that could be generated using the information available in the table.

EXAMPLE 1

Table xx — Definition of SIMPLE_STRUCTURE Structure

Parameter	Type	Description
tag	TPM_ST	
value1	INT32	
value2	INT32	

EXAMPLE 2

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

EXAMPLE 3

```
bool SIMPLE_STRUCTURE_Unmarshal(SIMPLE_STRUCTURE *target, BYTE **buffer, INT32 *size)
{
   // If unmarshal of tag succeeds
    if(TPM_ST_Unmarshal((TPM_ST *)&(target->tag), buffer, size))
        // then unmarshal value1, and if that succeeds...
        if(INT32_Unmarshal((INT32 *)&(target->value1, buffer, size))
            // then return the results of unmarshaling values
            return(INT32_Unmarshal((INT32 *)&(target->value2, buffer, size))
    // if unmarshal of tag or value failed, return failure
    return FALSE;
}
```


A table may have a term in {}. This indicates that the table is conditionally compiled. It is commonly used when a table's inclusion is based on the implementation of a cryptographic algorithm. See, for example, Table 160 — Definition of (TPM_ALG_ID) {RSA} TPMI_ALG_RSA_DECRYPT Type, which is dependent on the RSA algorithm.

5.8 Conditional Types

An enumeration may contain an extended value indicated by "+" preceding the name in the "Value" column. This "+" indicates that this is a conditional value that may be allowed in certain situations.

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.

EXAMPLE 1

Table xx — Definition of (CARD_SUIT) TPMI_CARD_SUIT Type

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

When an interface type is used, a "+" will be appended to the type specification for the parameter when the conditional value is allowed. If no "+" is present, then the conditional value is not allowed.

EXAMPLE 2

Table xx — Definition of POKER_CARD Structure

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

EXAMPLE 3

Table xx — Definition of BRIDGE_CARD Structure

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

5.9 Unions

5.9.1 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:

1) union definition;
2) union instance; and
3) union selector definition.

5.9.2 Union Definition

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

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

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

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

EXAMPLE 1

Table xx — Definition of NUMBER_UNION Union

Parameter	Type	Selector	Description
a_byte	BYTE	BYTE_SELECT	
an_int	int	INT_SELECT	
a_float	float	FLOAT_SELECT	
+null		NULL_SELECT	the empty branch

EXAMPLE 2

```
// C-compatible version of the union defined in the table above
typedef union {
    BYTE            a_byte;
    int             an_int;
    float           a_float;
} NUMBER_UNION;
```

EXAMPLE 3


```
// Possible auto-generated code to unmarshal a union in Example 2 based on the
// input value of selector
bool 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;
}
```

A table may have a type with no selector. This is used when the first part of the structure for all union members is identical. This type is a programming convenience, allowing code to reference the common members without requiring a case statement to determine the specific structure. In object oriented programming terms, this type is a superclass and the types with selectors are subclasses.

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

EXAMPLE 1

Table xx — Definition of STRUCTURE_WITH_UNION Structure

Parameter	Type	Description
select	NUMBER_SELECT	a value indicating the type in *number*
[select] number	NUMBER_UNION	a union as shown in 5.9.2

EXAMPLE 2

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

EXAMPLE 3

```
// Possible unmarshaling code for the structure above
bool STRUCT_WITH_UNION_Unmarshal(STRUCT_WITH_UNION *target, BYTE **buffer, INT32 *size)
{
    // Unmarshal the selector value
    if(!NUMBER_SELECT_Unmarshal((NUMBER_SELECT *)&target->select, buffer, size))
        return FALSE;
    // 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);
}
```


5.9.4 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 5.9.2, a selector definition should be limited to no more than four values, one for each of the union members. The selector definition could have fewer than four values.

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

EXAMPLE 1

Table xx — Definition of (INT8) NUMBER_SELECT Values <IN>

Name	Value	Comments
BYTE_SELECT	3	
INT_SELECT	2	
FLOAT_SELECT	1	
NULL_SELECT	0	

The unmarshaling code would limit the input values to the defined values. When the NUMBER_SELECT is used in the union instance of 5.9.3, any of the allowed union members of NUMBER_UNION could be present.

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

EXAMPLE 2

Table xx — Definition of (NUMBER_SELECT) AN_INTEGER Type <IN>

Values	Comments
{BYTE_SELECT, INT_SELECT}	list of allowed values

NOTE Since NULL_SELECT is not in the list of values accepted as a JUST_INTEGER, the “+” modifier will have no effect if used for a JUST_INTEGERS type shown in Example 3.

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

EXAMPLE 3

Table xx — Definition of JUST_INTEGERS Structure

Parameter	Type	Description
select	AN_INTEGER	a value indicating the type in *number*
[select] number	NUMBER_UNION	a union as shown in 5.9.2

5.10 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 Example 1 shows how a structure containing bit fields would be defined in ISO/IEC 11889. Example 2 shows the C code that might be produced from that table by an automated process.

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

EXAMPLE1

Table xx — Definition of (UINT32) SOME_ATTRIBUTE Bits

Bit	Name	Action
0	zeroth_bit	**SET (1)**: what to do if bit is 1 **CLEAR (0):** what to do if bit is 0
1	first_bit	**SET (1)**: what to do if bit is 1 **CLEAR (0):** what to do if bit is 0
6:2	Reserved	A placeholder that spans 5 bits
7	third_bit	**SET (1)**: what to do if bit is 1 **CLEAR (0):** what to do if bit is 0
31:8	Reserved	Placeholder to fill 32 bits

EXAMPLE 2

```
/* C language equivalent of the attributes structure defined in the table above */
typedef struct {
    int zeroth_bit : 1;
    int first_bit  : 1;
    int Reserved3  : 5;
    int third_bit  : 1;
    int Reserved7  : 24;
} SOME_ATTRIBUTE;
```

5.11 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.

Parameter limits expressed using braces apply only to inputs to the TPM. Any value returned by the TPM is assumed to be valid.

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

EXAMPLE

Table xx — Definition of B_STRUCT Structure

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

5.12 Enumeration Macro

An enumeration can be a list of allowed numeric values.

EXAMPLE The permitted sizes for an AES key might be expressed as in Table 123 — Definition of {AES} (TPM_KEY_BITS) TPMI_AES_KEY_BITS Type.

$AES_KEY_SIZE_BITS is a macro that will take the value of AES_KEY_SIZE_BITS from Table B.7 — Defines for AES Algorithm Constants and substitute it.

5.13 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 5.11, *value4* indicates how many octets to unmarshal for *array2*. This semantic applies when the size field determines the number of octets to unmarshal. However, in some cases, the subsequent structure is self-defining. If the size precedes a parameter that is not an octet array, then the unmarshaled size of that parameter is determined by its data type. The table in Example 1 shows a structure where the size parameter would nominally indicate the number of octets in the remainder of the structure.

EXAMPLE 1

Table xx — Definition of C_STRUCT Structure

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

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

In Example 2 below, the *size*= causes the parser to generate validation code that will check that the unmarshaled size of *someStructure* and *someData* adds to the value unmarshaled for *size*. When the "=" decoration is present, a value of zero is not allowed for the size.

EXAMPLE 2

Table xx — Definition of D_STRUCT Structure

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

5.14 Data Direction

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

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

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

Table xx — Definition of TPMS_A Structure

Table xx — Definition of TPMS_A Structure <IN/OUT>

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

Table xx — Definition of TPMS_A Structure <IN>

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

Table xx — Definition of TPMS_A Structure <OUT>

5.15 Structure Validations

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

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

Table xx — Definition of TPMT_A Structure <S>

Table xx — Definition of TPMT_A Structure <IN, S>

Table xx — Definition of TPMT_A Structure <IN/OUT, S>

5.16 Name Prefix Convention

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

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

Table 1 — Name Prefix Convention

Prefix	Description
TPM	an indication/signal from the TPM’s system interface
TPM_	a constant or an enumerated type
TPM2_	a command defined by ISO/IEC 11889
TPM2B_	a structure that is a sized buffer where the size of the buffer is contained in a 16-bit, unsigned value The first parameter is the size in octets of the second parameter. The second parameter may be any type.
TPMA_	a structure where each of the fields defines an attribute and each field is usually a single bit All the attributes in an attribute structure are packed with the overall size of the structure indicated in the heading of the attribute description (UINT8, UINT16, or UINT32).
TPM_ALG_	an enumerated type that indicates an algorithm A TPM_ALG_ is often used as a selector for a union.
TPMI_	an interface type The value is specified for purposes of dynamic type checking when unmarshaled.
TPML_	a list length followed by the indicated number of entries of the indicated type This is an array with a length field.
TPMS_	a structure that is not a size buffer or a tagged buffer or a list
TPMT_	a structure with the first parameter being a structure tag, indicating the type of the structure that follows A structure tag may be either a TPMT_ST_ or TPM_ALG_ depending on context.
TPMU_	a union of structures, lists, or unions If a union exists, there will normally be a companion TPMT_ that is the expression of the union in a tagged structure, where the tag is the selector indicating which member of the union is present.
TPM_xx_	an enumeration value of a particular type

	The value of “xx” will be indicative of the use of the enumerated type. A table of “TPM_xx” constant definitions will exist to define each of the TPM_xx_ values.
EXAMPLE 1	Regarding the prefix TPM_xx_, TPM_CC_ indicates that the type is used for a *commandCode.* The allowed enumeration values will be found in the table defining the TPM_CC constants (Table 12).
EXAMPLE 2	Regarding the prefix TPM_xx_, TPM_RC_ indicates that the type is used for a *responseCode*. The allowed enumeration values are in Table 16.

5.17 Data Alignment

The data structures in this part of ISO/IEC 11889 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.

5.18 Parameter Unmarshaling Errors

The TPM commands are defined in ISO/IEC 11889-3. The command definition includes C code that details the actions performed by that command. The code is written assuming that the parameters of the command have been unmarshaled.

NOTE 1 An implementation does not need to process parameters in this manner or to separate the parameter parsing from the command actions. This method was chosen for ISO/IEC 11889 so that the normative behavior described by the detailed actions would be clear and unencumbered.

Unmarshaling is the process of processing the parameters in the input buffer and preparing the parameters for use by the command-specific action code. No data movement need take place but it is required that the TPM validate that the parameters meet the requirements of the expected data type as defined in this part of ISO/IEC 11889.

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.

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

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

Table 2 — Unmarshaling Errors

Response code	Usage
TPM_RC_INSUFFICIENT	the input buffer did not contain enough octets to allow unmarshaling of the expected data type;
TPM_RC_RESERVED	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 definition in this part of ISO/IEC 11889 of the parameter type even if that parameter is not used in the command actions.

6 Base Types

6.1 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 ISO/IEC 11889.

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

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

Table 3 — Definition of Base Types

Type	Name	Description
uint8_t	UINT8	unsigned, 8-bit integer
uint8_t	BYTE	unsigned 8-bit integer
int8_t	INT8	signed, 8-bit integer
int	BOOL	a bit in an **`int`** This is not used across the interface but is used in many places in the code. If the type were sent on the interface, it would have to have a type with a specific number of bytes.
uint16_t	UINT16	unsigned, 16-bit integer
int16_t	INT16	signed, 16-bit integer
uint32_t	UINT32	unsigned, 32-bit integer
int32_t	INT32	signed, 32-bit integer
uint64_t	UINT64	unsigned, 64-bit integer
int64_t	INT64	signed, 64-bit integer

6.2 Miscellaneous Types

These types are defined either for compatibility with ISO/IEC 11889 (first edition) or for clarity of ISO/IEC 11889.

Table 4 — Definition of Types for Documentation Clarity

Type	Name	Description
UINT32	TPM_ALGORITHM_ID	this is the ISO/IEC 11889 (first edition) compatible form of the TPM_ALG_ID
UINT32	TPM_MODIFIER_INDICATOR	
UINT32	TPM_AUTHORIZATION_SIZE	the *authorizationSize* parameter in a command
UINT32	TPM_PARAMETER_SIZE	the *parameterSizeset* parameter in a command
UINT16	TPM_KEY_SIZE	a key size in octets
UINT16	TPM_KEY_BITS	a key size in bits

7 Constants

7.1 TPM_SPEC (Specification Version Values)

These values are readable with TPM2_GetCapability().

NOTE This table will require editing when ISO/IEC 11889 is updated.

Table 5 — 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 ISO/IEC 11889
TPM_SPEC_VERSION	107	the version number of ISO/IEC 11889 (001.07* 100)
TPM_SPEC_YEAR	2014	the year of the version
TPM_SPEC_DAY_OF_YEAR	23	the day of the year (March 18, 20154)

7.2 TPM_GENERATED

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

Table 6 — Definition of (UINT32) TPM_GENERATED Constants <O>

Name	Value	Comments
TPM_GENERATED_VALUE	0xff544347	0xFF‘TCG’ (FF 54 43 47_{16})

7.3 TPM_ALG_ID

The TCG maintains a registry of all algorithms that have an assigned algorithm ID. That registry is the definitive list of algorithms that may be supported by a TPM. Use of the algorithm identifiers defined in the TCG Algorithm Registry is mandatory, making the TCG Algorithm Registry an indispensable reference for implementing this International Standard.

NOTE 1 Inclusion of an algorithm does NOT indicate that the necessary claims of the algorithm are available under reasonable and non-discriminatory (RAND) terms from a TCG member.

Table 8 is a copy of the TPM_ALG_ID constants table in the TCG Algorithm Registry, Revision 1.15. Table 8 is provided for illustrative purposes only.

An algorithm ID is often used like a tag to determine the type of a structure in a context-sensitive way. The values for TPM_ALG_ID shall be in the range of 00 00_{16} – 7F FF_{16}. Other structure tags will be in the range 80 00_{16} – FF FF_{16}.

NOTE 2 In ISO/IEC 11889 (first edition), these were defined as 32-bit constants. ISO/IEC 11889 limits the future size of the algorithm ID to 16 bits. The TPM_ALGORITHM_ID data type will continue to be a 32-bit number.

An algorithm shall not be assigned a value in the range 00 $C1_{16}$ – 00 $C6_{16}$ in order to prevent any overlap with the command structure tags used in ISO/IEC 11889 (first edition).

The implementation of some algorithms is dependent on the presence of other algorithms. When there is a dependency, the algorithm that is required is listed in column labeled "D" (dependent) in Table 8.

EXAMPLE Implementation of TPM_ALG_RSASSA needs the RSA algorithm be implemented.

TPM_ALG_KEYEDHASH and TPM_ALG_NULL are required of all TPM implementations.

Table 7 — Legend for TPM_ALG_ID Table

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

Table 8 — Definition of (UINT16) TPM_ALG_ID Constants <IN/OUT, S>

Algorithm Name	Value	Type	Dep	C	Reference	Comments
TPM_ALG_ERROR	0x0000					should not occur
TPM_ALG_FIRST	0x0001					marker value
TPM_ALG_RSA	0x0001	A O		S	IETF RFC 3447	the RSA algorithm
TPM_ALG_SHA	0x0004	H		S	ISO/IEC 10118-3	the SHA1 algorithm
TPM_ALG_SHA1	0x0004	H		S	ISO/IEC 10118-3	redefinition for documentation consistency
TPM_ALG_HMAC	0x0005	H X		S	ISO/IEC 9797-2	Hash Message Authentication Code (HMAC) algorithm
TPM_ALG_AES	0x0006	S		S	ISO/IEC 18033-3	the AES algorithm with various key sizes
TPM_ALG_MGF1	0x0007	H M		S	IEEE Std 1363™-2000 IEEE Std 1363a™-2004	hash-based mask-generation function
TPM_ALG_KEYEDHASH	0x0008	H O		S	ISO/IEC 11889	an object type that may use XOR for encryption or an HMAC for signing and may also refer to a data object that is neither signing nor encrypting
TPM_ALG_XOR	0x000A	H S		S	ISO/IEC 11889	the XOR encryption algorithm
TPM_ALG_SHA256	0x000B	H		S	ISO/IEC 10118-3	the SHA 256 algorithm
TPM_ALG_SHA384	0x000C	H		A	ISO/IEC 10118-3	the SHA 384 algorithm
TPM_ALG_SHA512	0x000D	H		A	ISO/IEC 10118-3	the SHA 512 algorithm
TPM_ALG_NULL	0x0010			S	ISO/IEC 11889	Null algorithm
TPM_ALG_SM3_256	0x0012	H		A	GM/T 0004-2012	SM3 hash algorithm
TPM_ALG_SM4	0x0013	S		A	GM/T 0002-2012	SM4 symmetric block cipher
TPM_ALG_RSASSA	0x0014	A X	RSA	S	IETF RFC 3447	a signature algorithm defined in section 8.2 (RSASSA-PKCS1-v1_5)
TPM_ALG_RSAES	0x0015	A E	RSA	S	IETF RFC 3447	a padding algorithm defined in section 7.2 (RSAES-PKCS1-v1_5)
TPM_ALG_RSAPSS	0x0016	A X	RSA	S	IETF RFC 3447	a signature algorithm defined in section 8.1 (RSASSA-PSS)
TPM_ALG_OAEP	0x0017	A E H	RSA	S	IETF RFC 3447	a padding algorithm defined in section 7.1 (RSAES_OAEP)
TPM_ALG_ECDSA	0x0018	A X	ECC	S	ISO/IEC 14888-3	signature algorithm using elliptic curve cryptography (ECC)
TPM_ALG_ECDH	0x0019	A M	ECC	S	NIST SP800-56A	secret sharing using ECC Based on context, this can be either One-Pass Diffie-Hellman, C(1, 1, ECC CDH) defined in 6.2.2.2 or Full Unified Model C(2, 2, ECC CDH) defined in

Algorithm Name	Value	Type	Dep	C	Reference	Comments
						6.1.1.2
TPM_ALG_ECDAA	0x001A	A X N	ECC	S	ISO/IEC 11889	elliptic-curve based, anonymous signing scheme
TPM_ALG_SM2	0x001B	A X	ECC	A	GM/T 0003.1–2012 GM/T 0003.2–2012 GM/T 0003.3–2012 GM/T 0003.5–2012	SM2 – depending on context, either an elliptic-curve based, signature algorithm or a key exchange protocol
TPM_ALG_ECSCHNORR	0x001C	A X	ECC	S	ISO/IEC 11889	elliptic-curve based Schnorr signature
TPM_ALG_ECMQV	0x001D	A M	ECC	A	NIST SP800-56A	two-phase elliptic-curve key exchange – C(2, 2, ECC MQV) section 6.1.1.4
TPM_ALG_KDF1_SP800_56A	0x0020	H M	ECC	S	NIST SP800-56A	concatenation key derivation function (approved alternative 1) section 5.8.1
TPM_ALG_KDF2	0x0021	H M		A	IEEE Std 1363a-2004	key derivation function KDF2 section 13.2
TPM_ALG_KDF1_SP800_108	0x0022	H M		S	NIST SP800-108	a key derivation method Section 5.1 KDF in Counter Mode
TPM_ALG_ECC	0x0023	A O		S	ISO/IEC 15946-1	prime field ECC
TPM_ALG_SYMCIPHER	0x0025	O S		S	ISO/IEC 11889	the object type for a symmetric block cipher
TPM_ALG_CAMELLIA	0x0026	S		A	ISO/IEC 18033-3	Camellia is symmetric block cipher. The Camellia algorithm with various key sizes.
TPM_ALG_CTR	0x0040	S E		A	ISO/IEC 10116	Counter mode – if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode.
TPM_ALG_OFB	0x0041	S E		A	ISO/IEC 10116	Output Feedback mode – if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode.
TPM_ALG_CBC	0x0042	S E		A	ISO/IEC 10116	Cipher Block Chaining mode – if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode.
TPM_ALG_CFB	0x0043	S E		A	ISO/IEC 10116	Cipher Feedback mode – if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode.

Algorithm Name	Value	Type	Dep	C	Reference	Comments
TPM_ALG_ECB	0x0044	S E		A	ISO/IEC 10116	Electronic Codebook mode – if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode.
TPM_ALG_LAST	0x0044					marker value
reserved	0x00C1 through 0x00C6					0x00C1 – 0x00C6 are reserved to prevent any overlap with the command structure tags used in ISO/IEC 11889 (first edition)
reserved	0x8000 through 0xFFFF					reserved for other structure tags

NOTE 1 For TPM_ALG_SM2, the Type is listed as signing but, other uses are allowed according to context.

NOTE 2 For TPM_ALG_ECB, this mode is not recommended for uses unless the key is frequently rotated such as in video codecs.

7.4 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 Algorithm Registry, Revision 1.15. Table 9 is provided for illustrative purposes only.

Table 9 — Definition of (UINT16) {ECC} TPM_ECC_CURVE Constants <IN/OUT, S>

Name	Value	Classification	Comments
TPM_ECC_NONE	0x0000	Assigned	
TPM_ECC_NIST_P192	0x0001	Assigned	
TPM_ECC_NIST_P224	0x0002	Assigned	
TPM_ECC_NIST_P256	0x0003	TCG Standard	
TPM_ECC_NIST_P384	0x0004	Assigned	
TPM_ECC_NIST_P521	0x0005	Assigned	
TPM_ECC_BN_P256	0x0010	TCG Standard	curve to support ECDAA
TPM_ECC_BN_P638	0x0011	Assigned	curve to support ECDAA
TPM_ECC_SM2_P256	0x0020	Assigned	
#TPM_RC_CURVE			

7.5 TPM_CC (Command Codes)

7.5.1 Format

A command is a 32-bit structure with fields assigned as shown in Figure 1.

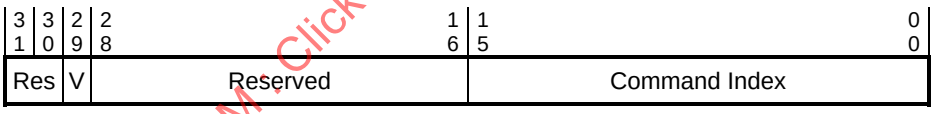

Figure 1 — Command Format

Table 10 — TPM Command Format Fields Description

Bit	Name	Definition
15:0	Command Index	the index of the command
28:16	Reserved	shall be zero
29	V	**SET(1):** the command is vendor specific **CLEAR(0):** the command is not vendor specific
31:30	Res	shall be zero

7.5.2 Description

Table 11 provides the legend for the interpretation of the column data in Table 12.

Table 11 — Legend for Command Code Tables

Column	Allowed Values	Comments
Name	Command Code Name	**Name** of the command
Command Code	Numeric value	the numeric value for the *commandCode*
NV Write	blank, Y, O	indicates whether the command may cause an NV write operation If this column contains a "Y," then successful completion of the command is expected to cause modification of the NV memory because of the command actions. If the column contains an "O," then the command may cause a modification to NV associated with an orderly shutdown. That is, the command may modify the orderly save state of NV, in which case, an NV write will be necessary. If the entry is blank, then writing to NV is not allowed in the command actions.
Physical Presence	blank, Y	indicates whether the Platform Authorization for this command may require confirmation through a physical presence indication
Decrypt	blank, 2, 4	A numeric value that indicates the number of octets in the size field of the first parameter of a command Blank indicates that no size field is present and no parameter encryption is allowed.
Encrypt	blank, 2, 4	A numeric value that indicates the number of octets in the size field of the first parameter of a response Blank indicates that no size field is present and no parameter encryption is allowed.
NOTE 1	Any command can be delayed in order for the TPM to complete NV actions due to a previous command or because of an asynchronous update of *Clock*.	
NOTE 2	Any command with an authorization value can cause an NV write on an authorization failure but the command does not complete successfully.	

7.5.3 TPM_CC Listing

Table 12 lists the command codes and their attributes. The only normative column in this table is the column indicating the command code assigned to a specific command (the "Command Code" column). For all other columns, the command and response tables in ISO/IEC 11889-3 are definitive.

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

Name	Command Code	NV Write	Physical Presence	Decrypt	Encrypt	Comments
TPM_CC_FIRST	0x0000011F					Compile variable. May decrease based on implementation.
TPM_CC_PP_FIRST	0x0000011F					Compile variable. Would decrease if new PP commands are added
TPM_CC_NV_UndefineSpaceSpecial	0x0000011F	Y	Y			
TPM_CC_EvictControl	0x00000120	Y	Y			
TPM_CC_HierarchyControl	0x00000121	Y	Y			
TPM_CC_NV_UndefineSpace	0x00000122	Y	Y			
TPM_CC_ChangeEPS	0x00000124	Y	Y			
TPM_CC_ChangePPS	0x00000125	Y	Y			
TPM_CC_Clear	0x00000126	Y	Y			
TPM_CC_ClearControl	0x00000127	Y	Y			
TPM_CC_ClockSet	0x00000128	Y	Y			
TPM_CC_HierarchyChangeAuth	0x00000129	Y	Y	2		
TPM_CC_NV_DefineSpace	0x0000012A	Y	Y	2		
TPM_CC_PCR_Allocate	0x0000012B	Y	Y			
TPM_CC_PCR_SetAuthPolicy	0x0000012C	Y	Y	2		
TPM_CC_PP_Commands	0x0000012D	Y	Y			
TPM_CC_SetPrimaryPolicy	0x0000012E	Y	Y	2		
TPM_CC_FieldUpgradeStart	0x0000012F	O	Y	2		
TPM_CC_ClockRateAdjust	0x00000130	O	Y			
TPM_CC_CreatePrimary	0x00000131		Y	2	2	
TPM_CC_NV_GlobalWriteLock	0x00000132	O	Y			
TPM_CC_PP_LAST	0x00000132					Compile variable
TPM_CC_GetCommandAuditDigest	0x00000133	Y		2		
TPM_CC_NV_Increment	0x00000134	Y				
TPM_CC_NV_SetBits	0x00000135	Y				
TPM_CC_NV_Extend	0x00000136	Y				
TPM_CC_NV_Write	0x00000137	Y		2		

Name	Command Code	NV Write	Physical Presence	Decrypt	Encrypt	Comments
TPM_CC_NV_WriteLock	0x00000138	Y				
TPM_CC_DictionaryAttackLockReset	0x00000139	O				
TPM_CC_DictionaryAttackParameters	0x0000013A	Y				
TPM_CC_NV_ChangeAuth	0x0000013B	Y		2		
TPM_CC_PCR_Event	0x0000013C	O		2		PCR
TPM_CC_PCR_Reset	0x0000013D	O				PCR
TPM_CC_SequenceComplete	0x0000013E	O		2	2	
TPM_CC_SetAlgorithmSet	0x0000013F	Y				
TPM_CC_SetCommandCodeAuditStatus	0x00000140	Y				
TPM_CC_FieldUpgradeData	0x00000141	O		2		
TPM_CC_IncrementalSelfTest	0x00000142	O				
TPM_CC_SelfTest	0x00000143	O				
TPM_CC_Startup	0x00000144	Y				
TPM_CC_Shutdown	0x00000145	Y				
TPM_CC_StirRandom	0x00000146	Y		2		
TPM_CC_ActivateCredential	0x00000147			2	2	
TPM_CC_Certify	0x00000148	O		2	2	
TPM_CC_PolicyNV	0x00000149			2		Policy
TPM_CC_CertifyCreation	0x0000014A	O		2	2	
TPM_CC_Duplicate	0x0000014B			2	2	
TPM_CC_GetTime	0x0000014C	O		2		
TPM_CC_GetSessionAuditDigest	0x0000014D	O		2		
TPM_CC_NV_Read	0x0000014E				2	
TPM_CC_NV_ReadLock	0x0000014F	O				
TPM_CC_ObjectChangeAuth	0x00000150			2	2	
TPM_CC_PolicySecret	0x00000151			2		Policy
TPM_CC_Rewrap	0x00000152			2	2	
TPM_CC_Create	0x00000153			2	2	
TPM_CC_ECDH_ZGen	0x00000154			2	2	
TPM_CC_HMAC	0x00000155			2	2	
TPM_CC_Import	0x00000156			2	2	
TPM_CC_Load	0x00000157			2	2	
TPM_CC_Quote	0x00000158	O		2	2	

Name	Command Code	NV Write	Physical Presence	Decrypt	Encrypt	Comments
TPM_CC_RSA_Decrypt	0x00000159				2	
TPM_CC_HMAC_Start	0x0000015B			2	2	
TPM_CC_SequenceUpdate	0x0000015C			2		
TPM_CC_Sign	0x0000015D			2		
TPM_CC_Unseal	0x0000015E				2	
TPM_CC_PolicySigned	0x00000160			2		Policy
TPM_CC_ContextLoad	0x00000161	O				Context
TPM_CC_ContextSave	0x00000162	O				Context
TPM_CC_ECDH_KeyGen	0x00000163				2	
TPM_CC_EncryptDecrypt	0x00000164				2	
TPM_CC_FlushContext	0x00000165	O				Context
TPM_CC_LoadExternal	0x00000167			2	2	
TPM_CC_MakeCredential	0x00000168			2	2	
TPM_CC_NV_ReadPublic	0x00000169					NV
TPM_CC_PolicyAuthorize	0x0000016A			2		Policy
TPM_CC_PolicyAuthValue	0x0000016B					Policy
TPM_CC_PolicyCommandCode	0x0000016C					Policy
TPM_CC_PolicyCounterTimer	0x0000016D			2		Policy
TPM_CC_PolicyCpHash	0x0000016E			2		Policy
TPM_CC_PolicyLocality	0x0000016F					Policy
TPM_CC_PolicyNameHash	0x00000170			2		Policy
TPM_CC_PolicyOR	0x00000171					Policy
TPM_CC_PolicyTicket	0x00000172			2		Policy
TPM_CC_ReadPublic	0x00000173				2	
TPM_CC_RSA_Encrypt	0x00000174			2	2	
TPM_CC_StartAuthSession	0x00000176	O		2	2	
TPM_CC_VerifySignature	0x00000177			2		
TPM_CC_ECC_Parameters	0x00000178					
TPM_CC_FirmwareRead	0x00000179					
TPM_CC_GetCapability	0x0000017A					
TPM_CC_GetRandom	0x0000017B				2	
TPM_CC_GetTestResult	0x0000017C					
TPM_CC_Hash	0x0000017D			2	2	

Name	Command Code	NV Write	Physical Presence	Decrypt	Encrypt	Comments
TPM_CC_PCR_Read	0x0000017E					PCR
TPM_CC_PolicyPCR	0x0000017F					Policy
TPM_CC_PolicyRestart	0x00000180					
TPM_CC_ReadClock	0x00000181					
TPM_CC_PCR_Extend	0x00000182	O		2		
TPM_CC_PCR_SetAuthValue	0x00000183	N		2		
TPM_CC_NV_Certify	0x00000184	O				
TPM_CC_EventSequenceComplete	0x00000185	O				
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	O		2	2	
TPM_CC_PolicyPassword	0x0000018C					Policy
TPM_CC_ZGen_2Phase	0x0000018D			2	2	
TPM_CC_EC_Ephemeral	0x0000018E					
TPM_CC_PolicyNvWritten	0x0000018F					Policy
TPM_CC_LAST	0x0000018F					Compile variable. May increase based on implementation.
#TPM_RC_COMMAND_CODE						

7.6 TPM_RC (Response Codes)

7.6.1 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 0 00 00_{16} and the low-order 12 bits (11:00) will contain the response code.

When a command succeeds, the TPM shall return TPM_RC_SUCCESS (0 00_{16}) 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 ISO/IEC 11889 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 ISO/IEC 11889. Commands that use tags defined in ISO/IEC 11889 (first edition) will use TPM_TAG_RSP_COMMAND in an error and a response code defined ISO/IEC 11889 (first edition).

If the tag of the command is not a recognized command tag, the TPM error response will differ depending on ISO/IEC 11889 (first edition) compatibility. If the TPM supports ISO/IEC 11889 (first edition) compatibility, the TPM shall return a tag of TPM_TAG_RSP_COMMAND and an appropriate ISO/IEC 11889 (first edition) response code (TPM_BADTAG = 00 00 00 $1E_{16}$). If the TPM does not support compatibility with ISO/IEC 11889 (first edition), 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.

7.6.2 Response Code Formats

The response codes for ISO/IEC 11889 are defined such that there is no overlap between the response codes used for ISO/IEC 11889 and those assigned in ISO/IEC 11889 (first edition).

The formats defined in clause 7.6.2 only apply when the tag for the response is TPM_ST_NO_SESSIONS.

The response codes use two different format groups. One group contains the ISO/IEC 11889 (first edition) compatible response codes and the response codes for ISO/IEC 11889 that are not related to command parameters. The second group contains the errors that may be associated with a command parameter, handle, or session.

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

bit	11	10	09	08	07	06	05	04	03	02	01	00
	S	T	r	V	F	E						

Figure 2 — Format-Zero Response Codes

The field definitions are:

Table 13 — Format-Zero Response Codes

Bit	Name	Definition
06:00	E	the error number The interpretation of this field is dependent on the setting of the F and S fields.
07	F	format selector **CLEAR** when the format is as defined in this Table 13 or when the response code is TPM_RC_BAD_TAG.
08	V	version **SET (1)**: The error number is defined in this part of ISO/IEC 11889 and is returned when the response tag is TPM_ST_NO_SESSIONS. **CLEAR (0)**: The error number is defined by ISO/IEC 11889 (first edition). The error number is returned when the response tag is TPM_TAG_RSP_COMMAND.
09	Reserved	shall be zero.
10	T	TCG/Vendor indicator **SET (1):** The response code is defined by the TPM vendor. **CLEAR (0):** The response code is defined by the TCG (a value in this part of ISO/IEC 11889).
11	S	severity **SET (1)**: The response code is a warning and the command was not necessarily in error. This command indicates that the TPM is busy or that the resources of the TPM have to be adjusted in order to allow the command to execute. **CLEAR (0)**: The response code indicates that the command had an error that would prevent it from running.
NOTE 1		In any error number returned by a TPM, the F (bit 7) and V (bit 8) attributes will be CLEAR when the response tag is TPM_TAG_RSP_COMMAND value used in ISO/IEC 11889 (first edition).
NOTE 2		The TCG/Vendor indicator attribute does not indicate a vendor-specific code unless the *F* attribute (bit[07]) is CLEAR.

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

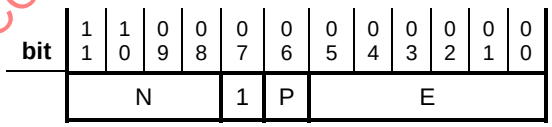

Figure 3 — Format-One Response Codes

There are 64 errors with this format. The errors can be associated with a parameter, handle, or session. The error number for this format is in bits[05:00]. When an error is associated with a parameter, 0 40_{16} is added and N is set to the parameter number.

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 14 — Format-One Response Codes

Bit	Name	Definition
05:00	E	the error number The error number is independent of the other settings.
06	P	**SET (1):** The error is associated with a parameter. **CLEAR (0):** The error is associated with a handle or a session.
07	F	the response code format selector This field shall be SET for the format in this table.
11:08	N	the number of the handle, session, or parameter in error 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 0000_2 and 0111_2. Sessions use values between 1000_2 and 1111_2.

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

Table 15 — Response Code Groupings

Bit			Definition
08	07	06	
0	0	x	a response code defined by ISO/IEC 11889 (first edition)
1	0	x	a response code defined by this part of ISO/IEC 11889 with no handle, session, or parameter number modifier
x	1	0	a response code defined by this part of ISO/IEC 11889 with either a handle or session number modifier
x	1	1	a response code defined by this part of ISO/IEC 11889 with a parameter number modifier
NOTE			An “x” in a column indicates that this may be either 0 or 1 and not affect the grouping of the response code.

7.6.3 TPM_RC Values

In general, response codes defined in this part of ISO/IEC 11889 will be unmarshaling errors and will have the F (format) bit SET. Codes that are unique to ISO/IEC 11889-3 will have the F bit CLEAR but the V (version) attribute will be SET to indicate that it is an ISO/IEC 11889 response code.

NOTE The constant RC_VER1 is used to indicate that the V attribute is SET and the constant RC_FMT1 is used to indicate that the F attribute is SET and that the return code is variable based on handle, session, and parameter modifiers.

Table 16 — Definition of (UINT32) TPM_RC Constants (Actions) <OUT>

Name	Value	Description
TPM_RC_SUCCESS	0x000	
TPM_RC_BAD_TAG	0x01E	defined for compatibility with ISO/IEC 11889 (first edition)
RC_VER1	0x100	set for all format 0 response codes
TPM_RC_INITIALIZE	RC_VER1 + 0x000	TPM not initialized
TPM_RC_FAILURE	RC_VER1 + 0x001	commands not being accepted because of a TPM failure
TPM_RC_SEQUENCE	RC_VER1 + 0x003	improper use of a sequence handle
TPM_RC_PRIVATE	RC_VER1 + 0x00B	
TPM_RC_HMAC	RC_VER1 + 0x019	
TPM_RC_DISABLED	RC_VER1 + 0x020	
TPM_RC_EXCLUSIVE	RC_VER1 + 0x021	command failed because audit sequence required exclusivity
TPM_RC_AUTH_TYPE	RC_VER1 + 0x024	authorization handle is not correct for command
TPM_RC_AUTH_MISSING	RC_VER1 + 0x025	command requires an authorization session for handle and it is not present.
TPM_RC_POLICY	RC_VER1 + 0x026	policy Failure In Math Operation or an invalid authPolicy value
TPM_RC_PCR	RC_VER1 + 0x027	PCR check fail
TPM_RC_PCR_CHANGED	RC_VER1 + 0x028	PCR have changed since checked.
TPM_RC_UPGRADE	RC_VER1 + 0x02D	for all commands other than TPM2_FieldUpgradeData(), this code indicates that the TPM is in field upgrade mode; for TPM2_FieldUpgradeData(), this code indicates that the TPM is not in field upgrade mode
TPM_RC_TOO_MANY_CONTEXTS	RC_VER1 + 0x02E	context ID counter is at maximum.
TPM_RC_AUTH_UNAVAILABLE	RC_VER1 + 0x02F	authValue or authPolicy is not available for selected entity.
TPM_RC_REBOOT	RC_VER1 + 0x030	a _TPM_Init and Startup(CLEAR) is required before the TPM can resume operation.
TPM_RC_UNBALANCED	RC_VER1 + 0x031	the protection algorithms (hash and symmetric) are not reasonably balanced. The digest size of the hash must be larger than the key size of the symmetric algorithm.

Name	Value	Description
TPM_RC_COMMAND_SIZE	RC_VER1 + 0x042	command *commandSize* value is inconsistent with contents of the command buffer; either the size is not the same as the octets loaded by the hardware interface layer or the value is not large enough to hold a command header
TPM_RC_COMMAND_CODE	RC_VER1 + 0x043	command code not supported
TPM_RC_AUTHSIZE	RC_VER1 + 0x044	the value of *authorizationSize* is out of range or the number of octets in the Authorization Area is greater than required
TPM_RC_AUTH_CONTEXT	RC_VER1 + 0x045	use of an authorization session with a context command
TPM_RC_NV_RANGE	RC_VER1 + 0x046	NV offset+size is out of range.
TPM_RC_NV_SIZE	RC_VER1 + 0x047	Requested allocation size is larger than allowed.
TPM_RC_NV_LOCKED	RC_VER1 + 0x048	NV access locked.
TPM_RC_NV_AUTHORIZATION	RC_VER1 + 0x049	NV access authorization fails in command actions (this failure does not affect lockout.action)
TPM_RC_NV_UNINITIALIZED	RC_VER1 + 0x04A	an NV Index is used before being initialized or the state saved by TPM2_Shutdown(STATE) could not be restored
TPM_RC_NV_SPACE	RC_VER1 + 0x04B	insufficient space for NV allocation
TPM_RC_NV_DEFINED	RC_VER1 + 0x04C	NV Index or persistend object already defined
TPM_RC_BAD_CONTEXT	RC_VER1 + 0x050	context in TPM2_ContextLoad() is not valid
TPM_RC_CPHASH	RC_VER1 + 0x051	cpHash value already set or not correct for use
TPM_RC_PARENT	RC_VER1 + 0x052	handle for parent is not a valid parent
TPM_RC_NEEDS_TEST	RC_VER1 + 0x053	some function needs testing.
TPM_RC_NO_RESULT	RC_VER1 + 0x054	returned when an internal function cannot process a request due to an unspecified problem. This code is usually related to invalid parameters that are not properly filtered by the input unmarshaling code.
TPM_RC_SENSITIVE	RC_VER1 + 0x055	the sensitive area did not unmarshal correctly after decryption – this code is used in lieu of the other unmarshaling errors so that an attacker cannot determine where the unmarshaling error occurred
RC_MAX_FM0	RC_VER1 + 0x07F	largest version 1 code that is not a warning
		New Subsection
RC_FMT1	0x080	This bit is SET in all format 1 response codes The codes in this group may have a value added to them to indicate the handle, session, or parameter to which they apply.
TPM_RC_ASYMMETRIC	RC_FMT1 + 0x001	asymmetric algorithm not supported or not correct
TPM_RC_ATTRIBUTES	RC_FMT1 + 0x002	inconsistent attributes
TPM_RC_HASH	RC_FMT1 + 0x003	hash algorithm not supported or not appropriate
TPM_RC_VALUE	RC_FMT1 + 0x004	value is out of range or is not correct for the context

Name	Value	Description
TPM_RC_HIERARCHY	RC_FMT1 + 0x005	hierarchy is not enabled or is not correct for the use
TPM_RC_KEY_SIZE	RC_FMT1 + 0x007	key size is not supported
TPM_RC_MGF	RC_FMT1 + 0x008	mask generation function not supported
TPM_RC_MODE	RC_FMT1 + 0x009	mode of operation not supported
TPM_RC_TYPE	RC_FMT1 + 0x00A	the type of the value is not appropriate for the use
TPM_RC_HANDLE	RC_FMT1 + 0x00B	the handle is not correct for the use
TPM_RC_KDF	RC_FMT1 + 0x00C	unsupported key derivation function or function not appropriate for use
TPM_RC_RANGE	RC_FMT1 + 0x00D	value was out of allowed range.
TPM_RC_AUTH_FAIL	RC_FMT1 + 0x00E	the authorization HMAC check failed and DA counter incremented
TPM_RC_NONCE	RC_FMT1 + 0x00F	invalid nonce size
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	authroization failure without DA implications
TPM_RC_EXPIRED	RC_FMT1 + 0x023	the policy has expired
TPM_RC_POLICY_CC	RC_FMT1 + 0x024	the *commandCode* in the policy is not the *commandCode* of the command or the command code in a policy command references a command that is not implemented
TPM_RC_BINDING	RC_FMT1 + 0x025	public and sensitive portions of an object are not cryptographically bound
TPM_RC_CURVE	RC_FMT1 + 0x026	curve not supported
TPM_RC_ECC_POINT	RC_FMT1 + 0x027	point is not on the required curve.
		New Subsection
RC_WARN	0x900	set for warning response codes

Name	Value	Description
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
TPM_RC_LOCALITY	RC_WARN + 0x007	bad locality
TPM_RC_YIELDED	RC_WARN + 0x008	the TPM has suspended operation on the command; forward progress was made and the command may be retried. See ISO/IEC 11889-1, clause 38, “Multi-tasking”
TPM_RC_CANCELED	RC_WARN + 0x009	the command was canceled
TPM_RC_TESTING	RC_WARN + 0x00A	TPM is performing self-tests
TPM_RC_REFERENCE_H0	RC_WARN + 0x010	the 1st handle in the handle area references a transient object or session that is not loaded
TPM_RC_REFERENCE_H1	RC_WARN + 0x011	the 2nd handle in the handle area references a transient object or session that is not loaded
TPM_RC_REFERENCE_H2	RC_WARN + 0x012	the 3rd handle in the handle area references a transient object or session that is not loaded
TPM_RC_REFERENCE_H3	RC_WARN + 0x013	the 4th handle in the handle area references a transient object or session that is not loaded
TPM_RC_REFERENCE_H4	RC_WARN + 0x014	the 5th handle in the handle area references a transient object or session that is not loaded
TPM_RC_REFERENCE_H5	RC_WARN + 0x015	the 6th handle in the handle area references a transient object or session that is not loaded
TPM_RC_REFERENCE_H6	RC_WARN + 0x016	the 7th handle in the handle area references a transient object or session that is not loaded
TPM_RC_REFERENCE_S0	RC_WARN + 0x018	the 1st authorization session handle references a session that is not loaded
TPM_RC_REFERENCE_S1	RC_WARN + 0x019	the 2nd authorization session handle references a session that is not loaded
TPM_RC_REFERENCE_S2	RC_WARN + 0x01A	the 3rd authorization session handle references a session that is not loaded
TPM_RC_REFERENCE_S3	RC_WARN + 0x01B	the 4th authorization session handle references a session that is not loaded
TPM_RC_REFERENCE_S4	RC_WARN + 0x01C	the 5th session handle references a session that is not loaded
TPM_RC_REFERENCE_S5	RC_WARN + 0x01D	the 6th session handle references a session that is not loaded
TPM_RC_REFERENCE_S6	RC_WARN + 0x01E	the 7th authorization session handle references a session that is not loaded
TPM_RC_NV_RATE	RC_WARN + 0x020	the TPM is rate-limiting accesses to prevent wearout of NV

Name	Value	Description
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

NOTE 1 TPM_RC_FAILURE can be returned by TPM2_GetTestResult() as the *testResult* parameter.

NOTE 2 TPM_RC_OBJECT_HANDLES cannot occur on the reference implementation.

NOTE 3 For the response code TPM_RC_OBJECT_HANDLES there is no reason why an implementation would implement a design that would deplete handle space. Platform specifications are encouraged to forbid it.

NOTE 4 TPM_RC_YIELDED cannot occur on the reference implementation.

7.7 TPM_CLOCK_ADJUST

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

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

Table 17 — Definition of (INT8) TPM_CLOCK_ADJUST Constants <IN>

Name	Value	Comments
TPM_CLOCK_COARSE_SLOWER	-3	Slow the *Clock* update rate by one coarse adjustment step.
TPM_CLOCK_MEDIUM_SLOWER	-2	Slow the *Clock* update rate by one medium adjustment step.
TPM_CLOCK_FINE_SLOWER	-1	Slow the *Clock* update rate by one fine adjustment step.
TPM_CLOCK_NO_CHANGE	0	No change to the *Clock* update rate.
TPM_CLOCK_FINE_FASTER	1	Speed the *Clock* update rate by one fine adjustment step.
TPM_CLOCK_MEDIUM_FASTER	2	Speed the *Clock* update rate by one medium adjustment step.
TPM_CLOCK_COARSE_FASTER	3	Speed the *Clock* update rate by one coarse adjustment step.
#TPM_RC_VALUE		

7.8 TPM_EO (EA Arithmetic Operands)

Table 18 — Definition of (UINT16) TPM_EO Constants <IN/OUT>

Operation Name	Value	Comments
TPM_EO_EQ	0x0000	A = B
TPM_EO_NEQ	0x0001	A ≠ B
TPM_EO_SIGNED_GT	0x0002	A > B signed
TPM_EO_UNSIGNED_GT	0x0003	A > B unsigned
TPM_EO_SIGNED_LT	0x0004	A < B signed
TPM_EO_UNSIGNED_LT	0x0005	A < B unsigned
TPM_EO_SIGNED_GE	0x0006	A ≥ B signed
TPM_EO_UNSIGNED_GE	0x0007	A ≥ B unsigned
TPM_EO_SIGNED_LE	0x0008	A ≤ B signed
TPM_EO_UNSIGNED_LE	0x0009	A ≤ B unsigned
TPM_EO_BITSET	0x000A	All bits SET in B are SET in A. ((A&B)=B)
TPM_EO_BITCLEAR	0x000B	All bits SET in B are CLEAR in A. ((A&B)=0)
#TPM_RC_VALUE		Response code returned when unmarshaling of this type fails

7.9 TPM_ST (Structure Tags)

Structure tags are used to disambiguate structures. They are 16-bit values with the most significant bit SET so that they do not overlap TPM_ALG_ID values. A single exception is made for the value associated with TPM_ST_RSP_COMMAND (0x00C4), which has the same value as the TPM_TAG_RSP_COMMAND tag from ISO/IEC 11889 (first edition). This value is used when the TPM is compatible with ISO/IEC 11889 (first edition) 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 part of ISO/IEC 11889 have parameters that are unions of other structures. That is, a parameter may be one of several structures. The parameter will have a selector value that indicates which of the options is actually present.

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

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

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

Name	Value	Comments
TPM_ST_RSP_COMMAND	0x00C4	*tag* value for a response; used when there is an error in the tag. This is also the value returned from a TPM implementing ISO/IEC 11889 (first edition) when an error occurs. This value is used in ISO/IEC 11889 because an error in the command tag may prevent determination of the family. When this tag is used in the response, the response code will be TPM_RC_BAD_TAG (0 $1E_{16}$), which has the same numeric value as the ISO/IEC 11889 (first edition) response code for TPM_BADTAG.
TPM_ST_NULL	0X8000	no structure type specified
TPM_ST_NO_SESSIONS	0x8001	*tag* value for a command/response for a command defined in ISO/IEC 11889; indicating that the command/response has no attached sessions and no *authorizationSize*/*parameterSize* value is present If the *responseCode* from the TPM is not TPM_RC_SUCCESS, then the response tag shall have this value.
TPM_ST_SESSIONS	0x8002	*tag* value for a command/response for a command defined in ISO/IEC 11889; indicating that the command/response has one or more attached sessions and the *authorizationSize*/*parameterSize* field is present
reserved	0x8003	When used between application software and the TPM resource manager, this tag indicates that the command has no sessions and the handles are using the Name format rather than the 32-bit handle format. Between the TRM and TPM, this tag would occur in a response from a TPM that overlaps the *tag* parameter of a request with the *tag* parameter of a response, when the response has no associated sessions.

Name	Value	Comments
reserved	0x8004	When used between application software and the TPM resource manager, this tag indicates that the command has sessions and the handles are using the Name format rather than the 32-bit handle format. Between the TRM and TPM, would occur in a response from a TPM that overlaps the *tag* parameter of a request with the *tag* parameter of a response, when the response has authorization sessions.
TPM_ST_ATTEST_NV	0x8014	tag for an attestation structure
TPM_ST_ATTEST_COMMAND_AUDIT	0x8015	tag for an attestation structure
TPM_ST_ATTEST_SESSION_AUDIT	0x8016	tag for an attestation structure
TPM_ST_ATTEST_CERTIFY	0x8017	tag for an attestation structure
TPM_ST_ATTEST_QUOTE	0x8018	tag for an attestation structure
TPM_ST_ATTEST_TIME	0x8019	tag for an attestation structure
TPM_ST_ATTEST_CREATION	0x801A	tag for an attestation structure
reserved	0x801B	do not use
TPM_ST_CREATION	0x8021	tag for a ticket type
TPM_ST_VERIFIED	0x8022	tag for a ticket type
TPM_ST_AUTH_SECRET	0x8023	tag for a ticket type
TPM_ST_HASHCHECK	0x8024	tag for a ticket type
TPM_ST_AUTH_SIGNED	0x8025	tag for a ticket type
TPM_ST_FU_MANIFEST	0x8029	tag for a structure describing a Field Upgrade Policy

NOTE 1 In a previously published version of the TCG TPM 2.0 Library specification, TPM_RC_BAD_TAG was incorrectly assigned a value of 0x030 instead of 30 (0x01e). Some implementations my return the old value instead of the new value.

NOTE 2 Regarding the value 0x8003, the response to application software will have a *tag* of TPM_ST_NO_SESSIONS.

NOTE 3 Regarding the value 0x8003, this value is not used by all TPM or TPM Resource Manager implementations.

NOTE 4 Regarding the value 0x8004, if the command completes successfully, the response to application software will have a *tag* of TPM_ST_SESSIONS.

NOTE 5 Regarding the value 0x8004, this value is not used by all TPM or TPM Resource Manager implementations.

NOTE 6 Regarding the value 0x801B, This was previously assigned to TPM_ST_ATTEST_NV. The tag is changed because the structure has changed.

7.10 TPM_SU (Startup Type)

These values are used in TPM2_Startup() to indicate the shutdown and startup mode. The defined startup sequences are:

a) TPM Reset – Two cases:

   1) Shutdown(CLEAR) followed by Startup(CLEAR)

   2) Startup(CLEAR) with no Shutdown()

b) TPM Restart – Shutdown(STATE) followed by Startup(CLEAR)

c) TPM Resume – Shutdown(STATE) followed by Startup(STATE)

TPM_SU values of 80 00_{16} 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 FF_{16} indicates that the startup state has not been set. If this was defined in this table to be, say, TPM_SU_NONE, then TPM_SU_NONE would be a valid input value but the caller is not allowed to indicate the that the startup type is TPM_SU_NONE so the reserved value is defined in the implementation as required for internal TPM uses.

Table 20 — Definition of (UINT16) TPM_SU Constants <IN>

Name	Value	Description
TPM_SU_CLEAR	0x0000	on TPM2_Shutdown(), indicates that the TPM should prepare for loss of power and save state required for an orderly startup (TPM Reset). on TPM2_Startup(), indicates that the TPM should perform TPM Reset or TPM Restart
TPM_SU_STATE	0x0001	on TPM2_Shutdown(), indicates that the TPM should prepare for loss of power and save state required for an orderly startup (TPM Restart or TPM Resume) on TPM2_Startup(), indicates that the TPM should restore the state saved by TPM2_Shutdown(TPM_SU_STATE)
#TPM_RC_VALUE		response code when incorrect value is used

7.11 TPM_SE (Session Type)

This type is used in TPM2_StartAuthSession() to indicate the type of the session to be created.

Table 21 — Definition of (UINT8) TPM_SE Constants <IN>

Name	Value	Description
TPM_SE_HMAC	0x00	
TPM_SE_POLICY	0x01	
TPM_SE_TRIAL	0x03	The policy session is being used to compute the *policyHash* and not for command authorization. This setting modifies some policy commands and prevents session from being used to authorize a command.
#TPM_RC_VALUE		response code when incorrect value is used

7.12 TPM_CAP (Capabilities)

The TPM_CAP values are used in TPM2_GetCapability() to select the type of the value to be returned. The format of the response varies according to the type of the value.

Table 22 — Definition of (UINT32) TPM_CAP Constants

Capability Name	Value	Property Type	Return Type
TPM_CAP_FIRST	0x00000000		
TPM_CAP_ALGS	0x00000000	TPM_ALG_ID(1)	TPML_ALG_PROPERTY
TPM_CAP_HANDLES	0x00000001	TPM_HANDLE	TPML_HANDLE
TPM_CAP_COMMANDS	0x00000002	TPM_CC	TPML_CCA
TPM_CAP_PP_COMMANDS	0x00000003	TPM_CC	TPML_CC
TPM_CAP_AUDIT_COMMANDS	0x00000004	TPM_CC	TPML_CC
TPM_CAP_PCRS	0x00000005	reserved	TPML_PCR_SELECTION
TPM_CAP_TPM_PROPERTIES	0x00000006	TPM_PT	TPML_TAGGED_TPM_PROPERTY
TPM_CAP_PCR_PROPERTIES	0x00000007	TPM_PT_PCR	TPML_TAGGED_PCR_PROPERTY
TPM_CAP_ECC_CURVES	0x00000008	TPM_ECC_CURVE(1)	TPML_ECC_CURVE
TPM_CAP_LAST	0x00000008		
TPM_CAP_VENDOR_PROPERTY	0x00000100	manufacturer specific	manufacturer-specific values
#TPM_RC_VALUE			

NOTE The TPM_ALG_ID or TPM_ECC_CURVE is cast to a UINT32.

7.13 TPM_PT (Property Tag)

The TPM_PT constants are used in TPM2_GetCapability(capability = TPM_CAP_TPM_PROPERTIES) to indicate the property being selected or returned.

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

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

Capability Name	Value	Comments
TPM_PT_NONE	0x00000000	indicates no property type
PT_GROUP	0x00000100	The number of properties in each group.
PT_FIXED	PT_GROUP * 1	the group of fixed properties returned as TPMS_TAGGED_PROPERTY The values in this group are only changed due to a firmware change in the TPM.
TPM_PT_FAMILY_INDICATOR	PT_FIXED + 0	a 4-octet character string containing the TPM Family value (TPM_SPEC_FAMILY)
TPM_PT_LEVEL	PT_FIXED + 1	the level of ISO/IEC 11889
TPM_PT_REVISION	PT_FIXED + 2	ISO/IEC 11889 Revision times 100
TPM_PT_DAY_OF_YEAR	PT_FIXED + 3	ISO/IEC 11889 publication day of year using TCG calendar
TPM_PT_YEAR	PT_FIXED + 4	ISO/IEC 11889 publication year using the CE
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
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 vendor-specific value indicating the version of the firmware
TPM_PT_FIRMWARE_VERSION_2	PT_FIXED + 12	the least-significant 32 bits of a vendor-specific value indicating the version of the firmware
TPM_PT_INPUT_BUFFER	PT_FIXED + 13	the maximum size of a parameter (typically, a TPM2B_MAX_BUFFER)
TPM_PT_HR_TRANSIENT_MIN	PT_FIXED + 14	the minimum number of transient objects that can be held in TPM RAM
TPM_PT_HR_PERSISTENT_MIN	PT_FIXED + 15	the minimum number of persistent objects that can be held in TPM NV memory
TPM_PT_HR_LOADED_MIN	PT_FIXED + 16	the minimum number of authorization sessions that can be held in TPM RAM
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.

Capability Name	Value	Comments
TPM_PT_PCR_COUNT	PT_FIXED + 18	the number of PCR implemented
TPM_PT_PCR_SELECT_MIN	PT_FIXED + 19	the minimum number of octets in a TPMS_PCR_SELECT.*sizeOfSelect*
TPM_PT_CONTEXT_GAP_MAX	PT_FIXED + 20	the maximum allowed difference (unsigned) between the *contextID* values of two saved session contexts This value shall be at least 2^{16}-1 (65535).
	PT_FIXED + 21	skipped
TPM_PT_NV_COUNTERS_MAX	PT_FIXED + 22	the maximum number of NV Indexes that are allowed to have the TPMA_NV_COUNTER attribute SET
TPM_PT_NV_INDEX_MAX	PT_FIXED + 23	the maximum size of an NV Index data area
TPM_PT_MEMORY	PT_FIXED + 24	a TPMA_MEMORY indicating the memory management method for the TPM
TPM_PT_CLOCK_UPDATE	PT_FIXED + 25	interval, in milliseconds, between updates to the copy of TPMS_CLOCK_INFO.*clock* in NV
TPM_PT_CONTEXT_HASH	PT_FIXED + 26	the algorithm used for the integrity HMAC on saved contexts and for hashing the *fuData* of TPM2_FirmwareRead()
TPM_PT_CONTEXT_SYM	PT_FIXED + 27	TPM_ALG_ID, the algorithm used for encryption of saved contexts
TPM_PT_CONTEXT_SYM_SIZE	PT_FIXED + 28	TPM_KEY_BITS, the size of the key used for encryption of saved contexts
TPM_PT_ORDERLY_COUNT	PT_FIXED + 29	the modulus - 1 of the count for NV update of an orderly counter The returned value is MAX_ORDERLY_COUNT. This will have a value of $2^N - 1$ where $1 \leq N \leq 32$
TPM_PT_MAX_COMMAND_SIZE	PT_FIXED + 30	the maximum value for *commandSize* in a command
TPM_PT_MAX_RESPONSE_SIZE	PT_FIXED + 31	the maximum value for *responseSize* in a response
TPM_PT_MAX_DIGEST	PT_FIXED + 32	the maximum size of a digest that can be produced by the TPM
TPM_PT_MAX_OBJECT_CONTEXT	PT_FIXED + 33	the maximum size of an object context that will be returned by TPM2_ContextSave
TPM_PT_MAX_SESSION_CONTEXT	PT_FIXED + 34	the maximum size of a session context that will be returned by TPM2_ContextSave
TPM_PT_PS_FAMILY_INDICATOR	PT_FIXED + 35	platform-specific family (a TPM_PS value)(see Table 25)
TPM_PT_PS_LEVEL	PT_FIXED + 36	the level of the platform-specific specification
TPM_PT_PS_REVISION	PT_FIXED + 37	the specification Revision times 100 for the platform-specific specification
TPM_PT_PS_DAY_OF_YEAR	PT_FIXED + 38	the platform-specific specification day of year using TCG calendar
TPM_PT_PS_YEAR	PT_FIXED + 39	the platform-specific specification year using the CE
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

Capability Name	Value	Comments
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 command
PT_VAR	PT_GROUP * 2	the group of variable properties returned as TPMS_TAGGED_PROPERTY The properties in this group change because of a Protected Capability other than a firmware update. The values are not necessarily persistent across all power transitions.
TPM_PT_PERMANENT	PT_VAR + 0	TPMA_PERMANENT
TPM_PT_STARTUP_CLEAR	PT_VAR + 1	TPMA_STARTUP_CLEAR
TPM_PT_HR_NV_INDEX	PT_VAR + 2	the number of NV Indexes currently defined
TPM_PT_HR_LOADED	PT_VAR + 3	the number of authorization sessions currently loaded into TPM RAM
TPM_PT_HR_LOADED_AVAIL	PT_VAR + 4	the number of additional authorization sessions, of any type, that could be loaded into TPM RAM This value is an estimate. If this value is at least 1, then at least one authorization session of any type may be loaded. Any command that changes the RAM memory allocation can make this estimate invalid.
TPM_PT_HR_ACTIVE	PT_VAR + 5	the number of active authorization sessions currently being tracked by the TPM This is the sum of the loaded and saved sessions.
TPM_PT_HR_ACTIVE_AVAIL	PT_VAR + 6	the number of additional authorization sessions, of any type, that could be created This value is an estimate. If this value is at least 1, then at least one authorization session of any type may be created. Any command that changes the RAM memory allocation can make this estimate invalid.
TPM_PT_HR_TRANSIENT_AVAIL	PT_VAR + 7	estimate of the number of additional transient objects that could be loaded into TPM RAM This value is an estimate. If this value is at least 1, then at least one object of any type may be loaded. Any command that changes the memory allocation can make this estimate invalid.
TPM_PT_HR_PERSISTENT	PT_VAR + 8	the number of persistent objects currently loaded into TPM NV memory
TPM_PT_HR_PERSISTENT_AVAIL	PT_VAR + 9	the number of additional persistent objects that could be loaded into NV memory This value is an estimate. If this value is at least 1, then at least one object of any type may be made persistent. Any command that changes the NV memory allocation can make this estimate invalid.
TPM_PT_NV_COUNTERS	PT_VAR + 10	the number of defined NV Indexes that have NV TPMA_NV_COUNTER attribute SET

Capability Name	Value	Comments
TPM_PT_NV_COUNTERS_AVAIL	PT_VAR + 11	the number of additional NV Indexes that can be defined with their TPMA_NV_COUNTER and TPMA_NV_ORDERLY attribute SET This value is an estimate. If this value is at least 1, then at least one NV Index may be created with the TPMA_NV_COUNTER and TPMA_NV_ORDERLY attributes SET. Any command that changes the NV memory allocation can make this estimate invalid.
TPM_PT_ALGORITHM_SET	PT_VAR + 12	code that limits the algorithms that may be used with the TPM
TPM_PT_LOADED_CURVES	PT_VAR + 13	the number of loaded ECC curves
TPM_PT_LOCKOUT_COUNTER	PT_VAR + 14	the current value of the lockout counter (*failedTries*)
TPM_PT_MAX_AUTH_FAIL	PT_VAR + 15	the number of authorization failures before DA lockout is invoked
TPM_PT_LOCKOUT_INTERVAL	PT_VAR + 16	the number of seconds before the value reported by TPM_PT_LOCKOUT_COUNTER is decremented
TPM_PT_LOCKOUT_RECOVERY	PT_VAR + 17	the number of seconds after a lockoutAuth failure before use of lockoutAuth may be attempted again
TPM_PT_NV_WRITE_RECOVERY	PT_VAR + 18	number of milliseconds before the TPM will accept another command that will modify NV This value is an approximation and may go up or down over time.
TPM_PT_AUDIT_COUNTER_0	PT_VAR + 19	the high-order 32 bits of the command audit counter
TPM_PT_AUDIT_COUNTER_1	PT_VAR + 20	the low-order 32 bits of the command audit counter

NOTE 1 Regarding PT_GROUP, the first group with any properties is group 1 (PT_GROUP * 1). Group 0 is reserved.

NOTE 2 Regarding TPM_PT_LEVEL, for this International Standard, the level is zero.

EXAMPLE 1 Regarding TPM_PT_REVISION, revision 01.01 would have a value of 101.

NOTE 3 Regarding TPM_PT_REVISION, for this International Standard, the Revision is 1.07.

EXAMPLE 2 Regarding TPM_PT_DAY_OF_YEAR, November 15, 2010, has a day of year value of 319 (00 00 01 $3F_{16}$).

NOTE 4 Regarding TPM_PT_DAY_OF_YEAR, the date is on the title page of this International Standard.

EXAMPLE 3 Regarding TPM_PT_YEAR, the year 2010 has a value of 00 00 07 DA_{16}.

NOTE 5 Regarding TPM_PT_YEAR, the date is on the title page of this International Standard.

NOTE 6 Regarding TPM_PT_VENDOR_STRING_1, 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.

NOTE 7 Regarding TPM_PT_HR_TRANSIENT_MIN, this minimum will be no less than the minimum value required by the platform-specific specification to which the TPM is built.

NOTE 8 Regarding TPM_PT_HR_PERSISTENT_MIN, this minimum will be no less than the minimum value required by the platform-specific specification to which the TPM is built.

NOTE 9 Regarding TPM_PT_HR_LOADED_MIN, this minimum will be no less than the minimum value required by the platform-specific specification to which the TPM is built.

Capability Name	Value	Comments
NOTE 10	Regarding TPM_PT_ACTIVE_SESSIONS_MAX, this value will be no less than the minimum value required by the platform-specific specification to which the TPM is built.	
NOTE 11	Regarding TPM_PT_PCR_COUNT, this number is determined by the defined attributes, not the number of PCR that are populated.	
NOTE 12	Regarding TPM_PT_PCR_SELECT_MIN, 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.	
NOTE 13	Regarding TPM_PT_NV_COUNTERS_MAX, it is possible for this value to be larger than the number of NV Indexes that can be defined. This would be indicative of a TPM implementation that did not use different implementation technology for different NV Index types.	
NOTE 14	Regarding TPM_PT_ORDERLY_COUNT, an “orderly counter” is an NV Index with TPMA_NV_COUNTER and TPMA_NV_ORDERLY both SET.	
NOTE 15	Regarding TPM_PT_ORDERLY_COUNT, when the low-order bits of a counter equal this value, an NV write occurs on the next increment.	
NOTE 16	Regarding TPM_PT_PS_FAMILY_INDICATOR, the platform-specific values for the TPM_PT_PS parameters are in the relevant platform-specific specification. In the reference implementation, all of these values are 0.	
NOTE 17	Regarding TPM_PT_HR_LOADED_AVAIL, a valid implementation might return 1 even if more than one authorization session would fit into RAM.	
NOTE 18	Regarding TPM_PT_HR_ACTIVE_AVAIL, a valid implementation might return 1 even if more than one authorization session could be created.	
NOTE 19	Regarding TPM_PT_HR_TRANSIENT_AVAIL, a valid implementation might return 1 even if more than one transient object would fit into RAM.	
NOTE 20	Regarding TPM_PT_HR_PERSISTENT_AVAIL, a valid implementation might return 1 even if more than one persistent object would fit into NV memory.	
NOTE 21	Regarding TPM_PT_NV_COUNTERS_AVAIL, a valid implementation might return 1 even if more than one NV counter could be defined.	

7.14 TPM_PT_PCR (PCR Property Tag)

The TPM_PT_PCR constants are used in TPM2_GetCapability() to indicate the property being selected or returned. The PCR properties can be read when *capability* == TPM_CAP_PCR_PROPERTIES.

Table 24 — Definition of (UINT32) TPM_PT_PCR Constants <IN/OUT, S>

Capability Name	Value	Comments
TPM_PT_PCR_FIRST	0x00000000	bottom of the range of TPM_PT_PCR properties
TPM_PT_PCR_SAVE	0x00000000	a SET bit in the TPMS_PCR_SELECT indicates that the PCR is saved and restored by TPM_SU_STATE
TPM_PT_PCR_EXTEND_L0	0x00000001	a SET bit in the TPMS_PCR_SELECT indicates that the PCR may be extended from locality 0 This property is only present if a locality other than 0 is implemented.
TPM_PT_PCR_RESET_L0	0x00000002	a SET bit in the TPMS_PCR_SELECT indicates that the PCR may be reset by TPM2_PCR_Reset() from locality 0
TPM_PT_PCR_EXTEND_L1	0x00000003	a SET bit in the TPMS_PCR_SELECT indicates that the PCR may be extended from locality 1 This property is only present if locality 1 is implemented.
TPM_PT_PCR_RESET_L1	0x00000004	a SET bit in the TPMS_PCR_SELECT indicates that the PCR may be reset by TPM2_PCR_Reset() from locality 1 This property is only present if locality 1 is implemented.
TPM_PT_PCR_EXTEND_L2	0x00000005	a SET bit in the TPMS_PCR_SELECT indicates that the PCR may be extended from locality 2 This property is only present if localities 1 and 2 are implemented.
TPM_PT_PCR_RESET_L2	0x00000006	a SET bit in the TPMS_PCR_SELECT indicates that the PCR may be reset by TPM2_PCR_Reset() from locality 2 This property is only present if localities 1 and 2 are implemented.
TPM_PT_PCR_EXTEND_L3	0x00000007	a SET bit in the TPMS_PCR_SELECT indicates that the PCR may be extended from locality 3 This property is only present if localities 1, 2, and 3 are implemented.
TPM_PT_PCR_RESET_L3	0x00000008	a SET bit in the TPMS_PCR_SELECT indicates that the PCR may be reset by TPM2_PCR_Reset() from locality 3 This property is only present if localities 1, 2, and 3 are implemented.
TPM_PT_PCR_EXTEND_L4	0x00000009	a SET bit in the TPMS_PCR_SELECT indicates that the PCR may be extended from locality 4 This property is only present if localities 1, 2, 3, and 4 are implemented.
TPM_PT_PCR_RESET_L4	0x0000000A	a SET bit in the TPMS_PCR_SELECT indicates that the PCR may be reset by TPM2_PCR_Reset() from locality 4 This property is only present if localities 1, 2, 3, and 4 are implemented.

Capability Name	Value	Comments
reserved	0x0000000B – 0x00000010	the values in this range are reserved They correspond to values that may be used to describe attributes associated with the extended localities (32-255).synthesize additional software localities. The meaning of these properties need not be the same as the meaning for the Extend and Reset properties above.
TPM_PT_PCR_NO_INCREMENT	0x00000011	a SET bit in the TPMS_PCR_SELECT indicates that modifications to this PCR (reset or Extend) will not increment the *pcrUpdateCounter*
TPM_PT_PCR_DRTM_RESET	0x00000012	a SET bit in the TPMS_PCR_SELECT indicates that the PCR is reset by a D-RTM event These PCR are reset to -1 on TPM2_Startup() and reset to 0 on a _TPM_Hash_End event following a _TPM_Hash_Start event.
TPM_PT_PCR_POLICY	0x00000013	a SET bit in the TPMS_PCR_SELECT indicates that the PCR is controlled by policy This property is only present if the TPM supports policy control of a PCR.
TPM_PT_PCR_AUTH	0x00000014	a SET bit in the TPMS_PCR_SELECT indicates that the PCR is controlled by an authorization value This property is only present if the TPM supports authorization control of a PCR.
reserved	0x00000015	reserved for the next (2nd) TPM_PT_PCR_POLICY set
reserved	0x00000016	reserved for the next (2nd) TPM_PT_PCR_AUTH set
reserved	0x00000017 – 0x00000210	reserved for the 2nd through 255th TPM_PT_PCR_POLICY and TPM_PT_PCR_AUTH values
reserved	0x00000211	reserved to the 256th, and highest allowed, TPM_PT_PCR_POLICY set
reserved	0x00000212	reserved to the 256th, and highest allowed, TPM_PT_PCR_AUTH set
reserved	0x00000213	new PCR property values may be assigned starting with this value
TPM_PT_PCR_LAST	0x00000014	top of the range of TPM_PT_PCR properties of the implementation If the TPM receives a request for a PCR property with a value larger than this, the TPM will return a zero length list and set the *moreData* parameter to NO.
NOTE Regarding TPM_PT_PCR_LAST, this is an implementation-specific value. The value shown reflects the reference code implementation.		

7.15 TPM_PS (Platform Specific)

The platform values in Table 25 are used for the TPM_PT_PS_FAMILY_INDICATOR.

Table 25 — Definition of (UINT32) TPM_PS Constants <OUT>

Capability Name	Value	Comments
TPM_PS_MAIN	0x00000000	not platform specific
TPM_PS_PC	0x00000001	PC Client
TPM_PS_PDA	0x00000002	PDA (includes all mobile devices that are not specifically cell phones)
TPM_PS_CELL_PHONE	0x00000003	Cell Phone
TPM_PS_SERVER	0x00000004	Server WG
TPM_PS_PERIPHERAL	0x00000005	Peripheral WG
TPM_PS_TSS	0x00000006	TSS WG
TPM_PS_STORAGE	0x00000007	Storage WG
TPM_PS_AUTHENTICATION	0x00000008	Authentication WG
TPM_PS_EMBEDDED	0x00000009	Embedded WG
TPM_PS_HARDCOPY	0x0000000A	Hardcopy WG
TPM_PS_INFRASTRUCTURE	0x0000000B	Infrastructure WG
TPM_PS_VIRTUALIZATION	0x0000000C	Virtualization WG
TPM_PS_TNC	0x0000000D	Trusted Network Connect WG
TPM_PS_MULTI_TENANT	0x0000000E	Multi-tenant WG
TPM_PS_TC	0x0000000F	Technical Committee
NOTE Values below six (6) have the same values as the purview assignments in ISO/IEC 11889 (first edition).		

8 Handles

8.1 Introduction

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

Table 26 — Definition of Types for Handles

Type	Name	Description
UINT32	TPM_HANDLE	

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

8.2 TPM_HT (Handle Types)

The 32-bit handle 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 27.

Table 27 — Definition of (UINT8) TPM_HT Constants <S>

Name	Value	Comments
TPM_HT_PCR	0x00	**PCR** – consecutive numbers, starting at 0, that reference the PCR registers A platform-specific specification will set the minimum number of PCR and an implementation may have more.
TPM_HT_NV_INDEX	0x01	**NV Index** – assigned by the caller
TPM_HT_HMAC_SESSION	0x02	**HMAC Authorization Session** – assigned by the TPM when the session is created
TPM_HT_LOADED_SESSION	0x02	**Loaded Authorization Session** – used only in the context of TPM2_GetCapability This type references both loaded HMAC and loaded policy authorization sessions.
TPM_HT_POLICY_SESSION	0x03	**Policy Authorization Session** – assigned by the TPM when the session is created
TPM_HT_ACTIVE_SESSION	0x03	**Active Authorization Session** – used only in the context of TPM2_GetCapability This type references saved authorization session contexts for which the TPM is maintaining tracking information.
TPM_HT_PERMANENT	0x40	**Permanent Values** – assigned by this part of ISO/IEC 11889 in Table 28
TPM_HT_TRANSIENT	0x80	**Transient Objects** – assigned by the TPM when an object is loaded into transient-object memory or when a persistent object is converted to a transient object
TPM_HT_PERSISTENT	0x81	**Persistent Objects** – assigned by the TPM when a loaded transient object is made persistent

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

EXAMPLE 1 If a TPM is only able to hold 4 transient objects in internal memory, it might choose to assign handles to those objects with the values 80 00 00 00_{16} – 80 00 00 03_{16}.

When a transient object is converted to a persistent object (TPM2_EvictControl()), the TPM shall validate that the handle provided by the caller has an MSO of TPM_HT_PERSISTENT and that the handle is not already assigned to a persistent object.

A handle is assigned to a session when the session is started. The handle shall have an MSO equal to TPM_HT_SESSION and remain associated with that session until the session is closed or flushed. The TPM shall ensure that a session handle is only associated with one session at a time. When the session is loaded into the TPM using TPM2_LoadContext(), it will have the same handle each time it is loaded.

EXAMPLE 2 If a TPM is only able to track 64 active sessions at a time, it could number those sessions using the values xx 00 01 00_{16} – xx 00 01 $3F_{16}$ where xx is either 02_{16} or 03_{16} depending on the session type.

8.3 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 of 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 00_{16} to 81 7F FF FF_{16} inclusive and the TPM will return an error if Owner Authorization is used to attempt to assign a persistent handle outside of this range.

8.4 TPM_RH (Permanent Handles)

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

Table 28 — Definition of (TPM_HANDLE) TPM_RH Constants <S>

Name	Value	Type	Comments
TPM_RH_FIRST	0x40000000	R	
TPM_RH_SRK	0x40000000	R	not used
TPM_RH_OWNER	0x40000001	K, A, P	handle references the Storage Primary Seed (SPS), the *ownerAuth*, and the *ownerPolicy*
TPM_RH_REVOKE	0x40000002	R	not 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 *authValue*, and an Empty Policy *authPolicy*.
TPM_RH_UNASSIGNED	0x40000008	R	value reserved to the TPM to indicate a handle location that has not been initialized or assigned
TPM_RS_PW	0x40000009	S	authorization value used to indicate a password authorization session
TPM_RH_LOCKOUT	0x4000000A	A	references the authorization associated with the dictionary attack lockout reset
TPM_RH_ENDORSEMENT	0x4000000B	K, A, P	references the Endorsement Primary Seed (EPS), *endorsementAuth*, and *endorsementPolicy*
TPM_RH_PLATFORM	0x4000000C	K, A, P	references the Platform Primary Seed (PPS), *platformAuth*, and *platformPolicy*
TPM_RH_PLATFORM_NV	0x4000000D	C	for phEnableNV
TPM_RH_AUTH_00	0x40000010	A	Start of a range of authorization values that are vendor-specific. A TPM may support any of the values in this range as are needed for vendor-specific purposes. Disabled if ehEnable is CLEAR.
TPM_RH_AUTH_FF	0x4000010F	A	End of the range of vendor-specific authorization values.

Name	Value	Type	Comments
TPM_RH_LAST	0x4000010F	R	the top of the reserved handle area This is set to allow TPM2_GetCapability() to know where to stop. It may vary as implementations add to the permanent handle area.

Type definitions:

- **R** – a reserved value
- **K** – a Primary Seed
- **A** – an authorization value
- **P** – a policy value
- **S** – a session handle
- **C** - a control

NOTE 1 The handles TPM_RH_SRK, TPM_RH_REVOKE, TPM_RH_TRANSPORT, TPM_RH_OPERATOR, TPM_RH_ADMIN and TPM_RH_EK, are only used in a TPM that is compatible with ISO/IEC 11889 (first edition). It is not used in any command defined in this International Standard.

NOTE 2 Regarding the values a TPM supports for TPM_RH_AUTH_00, “any” includes “none”.

8.5 TPM_HC (Handle Value Constants)

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

These values, that indicate ranges, are informative and may be changed by an implementation as long as the values stay within the prescribed ranges for the handle type:

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_LAST

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

PERSISTENT_FIRST, PERSISTENT_LAST, PLATFORM_PERSISTENT, NV_INDEX_FIRST, NV_INDEX_LAST, PERMANENT_FIRST, PERMANENT_LAST

NOTE 1 PCR0 is architecturally intended to have a handle value of 0.

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

A similar approach is used for transient objects with the first assigned handle at the bottom of the range defined by TPM_HT_TRANSIENT and the top of the range determined by the implementation-dependent value of MAX_LOADED_OBJECTS.

The first assigned handle for evict objects is also at the bottom of the allowed range defined by TPM_HT_PERSISTENT and the top of the range determined by the implementation-dependent value of MAX_EVICT_OBJECTS.

NOTE 2 The values in Table 29 are intended to facilitate the process of making the handle larger than 32 bits in the future. It is intended that HR_MASK and HR_SHIFT are the only values that need change to resize the handle space.

Table 29 — Definition of (TPM_HANDLE) TPM_HC Constants <S>

Name	Value	Comments
HR_HANDLE_MASK	0x00FFFFFF	to mask off the HR
HR_RANGE_MASK	0xFF000000	to mask off the variable part
HR_SHIFT	24	
HR_PCR	(TPM_HT_PCR << HR_SHIFT)	
HR_HMAC_SESSION	(TPM_HT_HMAC_SESSION << HR_SHIFT)	
HR_POLICY_SESSION	(TPM_HT_POLICY_SESSION << HR_SHIFT)	
HR_TRANSIENT	(TPM_HT_TRANSIENT << HR_SHIFT)	
HR_PERSISTENT	(TPM_HT_PERSISTENT << HR_SHIFT)	
HR_NV_INDEX	(TPM_HT_NV_INDEX << HR_SHIFT)	
HR_PERMANENT	(TPM_HT_PERMANENT << HR_SHIFT)	
PCR_FIRST	(HR_PCR + 0)	first PCR
PCR_LAST	(PCR_FIRST + IMPLEMENTATION_PCR-1)	last PCR
HMAC_SESSION_FIRST	(HR_HMAC_SESSION + 0)	first HMAC session
HMAC_SESSION_LAST	(HMAC_SESSION_FIRST+MAX_ACTIVE_SESSIONS-1)	last HMAC session
LOADED_SESSION_FIRST	HMAC_SESSION_FIRST	used in GetCapability
LOADED_SESSION_LAST	HMAC_SESSION_LAST	used in GetCapability
POLICY_SESSION_FIRST	(HR_POLICY_SESSION + 0)	first policy session
POLICY_SESSION_LAST	(POLICY_SESSION_FIRST + MAX_ACTIVE_SESSIONS-1)	last policy session
TRANSIENT_FIRST	(HR_TRANSIENT + 0)	first transient object
ACTIVE_SESSION_FIRST	POLICY_SESSION_FIRST	used in GetCapability
ACTIVE_SESSION_LAST	POLICY_SESSION_LAST	used in GetCapability
TRANSIENT_LAST	(TRANSIENT_FIRST+MAX_LOADED_OBJECTS-1)	last transient object
PERSISTENT_FIRST	(HR_PERSISTENT + 0)	first persistent object
PERSISTENT_LAST	(PERSISTENT_FIRST + 0x00FFFFFF)	last persistent object
PLATFORM_PERSISTENT	(PERSISTENT_FIRST + 0x00800000)	first platform persistent object
NV_INDEX_FIRST	(HR_NV_INDEX + 0)	first allowed NV Index
NV_INDEX_LAST	(NV_INDEX_FIRST + 0x00FFFFFF)	last allowed NV Index
PERMANENT_FIRST	TPM_RH_FIRST	
PERMANENT_LAST	TPM_RH_LAST	

9 Attribute Structures

9.1 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.

9.2 TPMA_ALGORITHM

This structure defines the attributes of an algorithm.

Each algorithm has a fundamental attribute: *asymmetric*, *symmetric*, or *hash*. In some cases (e.g., TPM_ALG_RSA or TPM_ALG_AES), this is the only attribute.

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

Table 30 — Definition of (UINT32) TPMA_ALGORITHM Bits

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

9.3 TPMA_OBJECT (Object Attributes)

9.3.1 Introduction

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

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

9.3.2 Structure Definition

Table 31 — Definition of (UINT32) TPMA_OBJECT Bits

Bit	Name	Definition
0	Reserved	shall be zero
1	fixedTPM	**SET (1):** The hierarchy of the object, as indicated by its Qualified Name, may not change. **CLEAR (0):** The hierarchy of the object may change as a result of this object or an ancestor key being duplicated for use in another hierarchy.
2	stClear	**SET (1):** Previously saved contexts of this object may not be loaded after Startup(CLEAR). **CLEAR (0):** Saved contexts of this object may be used after a Shutdown(STATE) and subsequent Startup().
3	Reserved	shall be zero
4	fixedParent	**SET (1):** The parent of the object may not change. **CLEAR (0):** The parent of the object may change as the result of a TPM2_Duplicate() of the object.
5	sensitiveDataOrigin	**SET (1):** Indicates that, when the object was created with TPM2_Create() or TPM2_CreatePrimary(), the TPM generated all of the sensitive data other than the *authValue*. **CLEAR (0):** A portion of the sensitive data, other than the *authValue*, was provided by the caller.
6	userWithAuth	**SET (1):** Approval of USER role actions with this object may be with an HMAC session or with a password using the *authValue* of the object or a policy session. **CLEAR (0):** Approval of USER role actions with this object may only be done with a policy session.
7	adminWithPolicy	**SET (1):** Approval of ADMIN role actions with this object may only be done with a policy session. **CLEAR (0):** Approval of ADMIN role actions with this object may be with an HMAC session or with a password using the *authValue* of the object or a policy session.
9:8	Reserved	shall be zero
10	noDA	**SET (1):** The object is not subject to dictionary attack protections. **CLEAR (0):** The object is subject to dictionary attack protections.
11	encryptedDuplication	**SET (1):** If the object is duplicated, then *symmetricAlg* shall not be TPM_ALG_NULL and *newParentHandle* shall not be TPM_RH_NULL. **CLEAR (0):** The object may be duplicated without an inner wrapper on the private portion of the object and the new parent may be TPM_RH_NULL.
15:12	Reserved	shall be zero
16	restricted	**SET (1):** Key usage is restricted to manipulate structures of known format; the parent of this key shall have *restricted* SET. **CLEAR (0):** Key usage is not restricted to use on special formats.

Bit	Name	Definition
17	decrypt	**SET (1):** The private portion of the key may be used to decrypt. **CLEAR (0):** The private portion of the key may not be used to decrypt.
18	sign	**SET (1):** The private portion of the key may be used to sign. **CLEAR (0):** The private portion of the key may not be used to sign.
31:19	Reserved	shall be zero

9.3.3 Attribute Descriptions

9.3.3.1 Introduction

The following remaining paragraphs in clause 9.3.3 describe the use and settings for each of the TPMA_OBJECT attributes. The description includes checks that are performed on the *objectAttributes* when an object is created, when it is loaded, and when it is imported. In these descriptions:

Creation – indicates settings for the *template* parameter in TPM2_Create() or TPM2_CreatePrimary()

Load – indicates settings for the *inPublic* parameter in TPM2_Load()

Import – indicates settings for the *objectPublic* parameter in TPM2_Import()

External – indicates settings that apply to the *inPublic* parameter in TPM2_LoadExternal() if both the public and sensitive portions of the object are loaded

NOTE For TPM2_LoadExternal() when only the public portion of the object is loaded, the only attribute checks are the checks in the validation code following Table 31 and the reserved attributes check.

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

9.3.3.2 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.

NOTE 1 This attribute is the logical inverse of the migratable attribute in ISO/IEC 11889 (first edition). That is, when this attribute is CLEAR, it is the equivalent to an ISO/IEC 11889 (first edition) object with migratable SET.

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

NOTE 2 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

9.3.3.3 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

9.3.3.4 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

9.3.3.5 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 1 If the *fixedTPM* attribute is SET, then this attribute is authoritative and accurately reflects the source of the sensitive area data. If the *fixedTPM* attribute is CLEAR, then validation of this attribute requires evaluation of the properties of the ancestor keys.

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

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

NOTE 3 The *inSensitive.sensitive.data.size* parameter might not be zero for a data object so *sensitiveDataOrigin* needs 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

9.3.3.6 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 1 Regardless of the setting of this attribute, authorizations for operations that require USER role authorizations can be provided with a policy session that satisfies the object's *authPolicy*.

NOTE 2 Regardless of the setting of this attribute, the *authValue* can be referenced in a policy session or used to provide the *bind* value in TPM2_StartAuthSession(). However, if *userWithAuth* is CLEAR, then the object can 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

9.3.3.7 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 then HMAC or password authorizations may not be used for ADMIN role authorizations.

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

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

Actions that always require policy (TPM2_Duplicate()) are not affected by the setting of this attribute.

Creation – may be SET or CLEAR in *template*

Load – may be SET or CLEAR

Import – may be SET or CLEAR

External – may be SET or CLEAR

9.3.3.8 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

9.3.3.9 Bit[11] – *encryptedDuplication*

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

NOTE 1 Enforcement of these requirements in TPM2_Duplicate() is by not allowing *symmetricAlg* to be TPM_ALG_NULL and not allowing *newParentHandle* to be TPM_RH_NULL.

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

NOTE 2 This requirement means that *encryptedDuplication* 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 3 This allows the object at the boundary between duplicable and non-duplicable objects to have either setting.

If an object's parent has *fixedTPM* CLEAR, then the object is required to have the same setting of *encryptedDuplication* as its parent.

NOTE 4 This requirement forces all duplicable objects in a duplication group to have the same *encryptedDuplication* setting.

Creation – shall be CLEAR if *fixedTPM* is SET. If *fixedTPM* is CLEAR, then this attribute shall have the same value as its parent unless *fixedTPM* is SET in the object's parent, in which case, it may be SET or CLEAR.

Load – shall be CLEAR if *fixedTPM* is SET. If *fixedTPM* is CLEAR, then this attribute shall have the same value as its parent, unless *fixedTPM* is SET the parent, in which case, it may be SET or CLEAR.

Import – if *fixedTPM* is SET in the object's new parent, then this attribute may be SET or CLEAR, otherwise, it shall have the same setting as the new parent.

External – may be SET or CLEAR.

9.3.3.10 Bit[16] – *restricted*

This 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 *template* if neither sign nor decrypt is SET in *template*.

Load – shall be CLEAR if neither sign nor decrypt is SET in the object

Import – may be SET or CLEAR

External – shall be CLEAR

9.3.3.11 Bit[17] – *decrypt*

When SET, the private portion of this key can be used to decrypt an external blob. If *restricted* is SET, then the TPM will return an error if the external decrypted blob is not formatted as appropriate for the command.

NOTE 1 Since TPM-generated keys and sealed data will contain a hash and a structure tag, the TPM can ensure that it is not being used to improperly decrypt and return sensitive data that ought not be returned. The only type of data that can be returned after decryption is a Sealed Data Object (a *keyedHash* object with *decrypt* and *sign* CLEAR).

When *restricted* is CLEAR, there are no restrictions on the use of the private portion of the key for decryption and the key may be used to decrypt and return any structure encrypted by the public portion of the key.

NOTE 2 A key with this attribute SET 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 *template*

Load – may be SET or CLEAR

Import – may be SET or CLEAR

External – may be SET or CLEAR

9.3.3.12 Bit[18] – *sign*

When this attribute is SET, the private portion of this key may be used to sign a digest. If *restricted* is SET, then the key may only be used to sign a digest that was computed by the TPM. A restricted signing key may be used to sign a TPM-generated digest. 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, whether generated by the TPM or externally provided.

NOTE 1 Some asymmetric algorithms might not support both *sign* and *decrypt* being SET in the same key.

If *sign* is SET on an object with *type* set to TPM_ALG_KEYEDHASH, it indicates that the object is an HMAC key.

NOTE 2 A key with this attribute SET cannot be a parent for another object.

Creation – shall not be SET if *decrypt* and *restricted* are both SET

Load – shall not be SET if *decrypt* and *restricted* are both SET

Import – shall not be SET if *decrypt* and *restricted* are both SET

External – shall not be SET if *decrypt* and *restricted* are both SET

9.4 TPMA_SESSION (Session Attributes)

This octet in each session is used to identify the session type, indicate its relationship to any handles in the command, and indicate its use in parameter encryption.

Table 32 — Definition of (UINT8) TPMA_SESSION Bits <IN/OUT>

Bit	Name	Meaning
0	continueSession	**SET (1):** In a command, this setting indicates that the session is to remain active after successful completion of the command. In a response, it indicates that the session is still active. If SET in the command, this attribute shall be SET in the response. **CLEAR (0):** In a command, this setting indicates that the TPM should close the session and flush any related context when the command completes successfully. In a response, it indicates that the session is closed and the context is no longer active. This attribute has no meaning for a password authorization and the TPM will allow any setting of the attribute in the command and SET the attribute in the response. This attribute will only be CLEAR in one response for a logical session. If the attribute is CLEAR, the context associated with the session is no longer in use and the space is available. A session created after another session is ended may have the same handle but logically is not the same session. This attribute has no effect if the command does not complete successfully.
1	auditExclusive	**SET (1):** In a command, this setting indicates that the command should only be executed if the session is exclusive at the start of the command. In a response, it indicates that the session is exclusive. This setting is only allowed if the *audit* attribute is SET. **CLEAR (0):** If *audit* is CLEAR, then this field is reserved but the error is TPM_RC_ATTRIBUTES rather than TPM_RC_RESERVED_BITS. See ISO/IEC 11889-1, clause 20.2, "Exclusive Audit Sessions".
2	auditReset	**SET (1):** In a command, this setting indicates that the audit digest of the session should be initialized and the exclusive status of the session SET. **CLEAR (0):** If *audit* is CLEAR, then this field is reserved but the error is TPM_RC_ATTRIBUTES rather than TPM_RC_RESERVED_BITS. This setting is always used for a response.
4:3	Reserved	shall be CLEAR
5	decrypt	**SET (1):** In a command, this setting indicates that the first parameter in the command is symmetrically encrypted using the parameter encryption scheme specified in ISO/IEC 11889-1. The TPM will decrypt the parameter after performing any HMAC computations and before unmarshaling the parameter. In a response, the attribute is copied from the request but has no effect on the response. **CLEAR (0):** Session not used for encryption. For a password authorization, this attribute will be CLEAR in both the command and response. This attribute may only be SET in one session per command. This attribute may be SET in a session that is not associated with a command handle. Such a session is provided for purposes of encrypting a parameter and not for authorization. This attribute may be SET in combination with any other session attributes. This attribute may only be SET if the first parameter of the command is a sized buffer (TPM2B_).

Bit	Name	Meaning
6	encrypt	**SET (1):** In a command, this setting indicates that the TPM should use this session to encrypt the first parameter in the response. In a response, it indicates that the attribute was set in the command and that the TPM used the session to encrypt the first parameter in the response using the parameter encryption scheme specified in ISO/IEC 11889-1. **CLEAR (0):** Session not used for encryption. For a password authorization, this attribute will be CLEAR in both the command and response. This attribute may only be SET in one session per command. This attribute may be SET in a session that is not associated with a command handle. Such a session is provided for purposes of encrypting a parameter and not for authorization. This attribute may only be SET if the first parameter of a response is a sized buffer (TPM2B_).
7	audit	**SET (1):** In a command or response, this setting indicates that the session is for audit and that *auditExclusive* and *auditReset* have meaning. This session may also be used for authorization, encryption, or decryption. The *encrypted* and *encrypt* fields may be SET or CLEAR. **CLEAR (0):** Session is not used for audit. This attribute may only be SET in one session per command or response. If SET in the command, then this attribute will be SET in the response.

9.5 TPMA_LOCALITY (Locality Attribute)

In a TPMS_CREATION_DATA structure, this structure is used to indicate the locality of the command that created the object. No more than one of the locality attributes shall be set in the creation data.

When used in TPM2_PolicyLocality(), this structure indicates which localities are approved by the policy. When a policy is started, all localities are allowed. If TPM2_PolicyLocality() is executed, it indicates that the command may only be executed at specific localities. More than one locality may be selected.

EXAMPLE 1 TPM_LOC_TWO would indicate that only locality 2 is authorized.

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

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

EXAMPLE 4 A value of 21_{16} would represent a locality of 33.

NOTE Locality values of 5 through 31 are not selectable.

If Extended is non-zero, then an extended locality is indicated and the TPMA_LOCALITY contains an integer value.

Table 33 — Definition of (UINT8) TPMA_LOCALITY Bits <IN/OUT>

Bit	Name	Definition
0	TPM_LOC_ZERO	
1	TPM_LOC_ONE	
2	TPM_LOC_TWO	
3	TPM_LOC_THREE	
4	TPM_LOC_FOUR	
7:5	Extended	If any of these bits is set, an extended locality is indicated

9.6 TPMA_PERMANENT

The attributes in this structure are persistent and are not changed as a result of _TPM_Init or any TPM2_Startup(). Some of the attributes in this structure may change as the result of specific Protected Capabilities. This structure may be read using TPM2_GetCapability(*capability* = TPM_CAP_TPM_PROPERTIES, *property* = TPM_PT_PERMANENT).

Table 34 — Definition of (UINT32) TPMA_PERMANENT Bits <OUT>

Bit	Parameter	Description
0	ownerAuthSet	**SET (1):** TPM2_HierarchyChangeAuth() with *ownerAuth* has been executed since the last TPM2_Clear(). **CLEAR (0):** *ownerAuth* has not been changed since TPM2_Clear().
1	endorsementAuthSet	**SET (1):** TPM2_HierarchyChangeAuth() with *endorsementAuth* has been executed since the last TPM2_Clear(). **CLEAR (0):** endorsementAuth has not been changed since TPM2_Clear().
2	lockoutAuthSet	**SET (1):** TPM2_HierarchyChangeAuth() with *lockoutAuth* has been executed since the last TPM2_Clear(). **CLEAR (0):** *lockoutAuth* has not been changed since TPM2_Clear().
7:3	Reserved	
8	disableClear	**SET (1):** TPM2_Clear() is disabled. **CLEAR (0):** TPM2_Clear() is enabled.
9	inLockout	**SET (1):** The TPM is in lockout and commands that require authorization with other than Platform Authorization or Lockout Authorization will not succeed.
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	
NOTE	See ISO/IEC 11889-3, clause 25.7, “TPM2_ClearControl” for details on changing the disableClear attribute.	

9.7 TPMA_STARTUP_CLEAR

These attributes are set to their default state on reset on each TPM Reset or TPM Restart. The attributes are preserved on TPM Resume.

On each TPM2_Startup(TPM_SU_CLEAR), the TPM will set these attributes to their indicated defaults.

This structure may be read using TPM2_GetCapability(*capability* = TPM_CAP_TPM_PROPERTIES, *property* = TPM_PT_STARTUP_CLEAR).

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

Table 35 — Definition of (UINT32) TPMA_STARTUP_CLEAR Bits <OUT>

Bit	Parameter	Description
0	phEnable	**SET (1):** The platform hierarchy is enabled and *platformAuth* or *platformPolicy* may be used for authorization. **CLEAR (0):** *platformAuth* and *platformPolicy* may not be used for authorizations, and objects in the platform hierarchy, including persistent objects, cannot be used.
1	shEnable	**SET (1):** The Storage hierarchy is enabled and *ownerAuth* or *ownerPolicy* may be used for authorization. NV indices defined using owner authorization are accessible. **CLEAR (0):** *ownerAuth* and *ownerPolicy* may not be used for authorizations, and objects in the Storage hierarchy, persistent objects, and NV indices defined using owner authorization cannot be used.
2	ehEnable	**SET (1):** The EPS hierarchy is enabled and Endorsement Authorization may be used to authorize commands. **CLEAR (0):** Endorsement Authorization may not be used for authorizations, and objects in the endorsement hierarchy, including persistent objects, cannot be used.
3	phEnableNV	**SET (1):** NV indices that have TPMA_PLATFORM_CREATE SET may be read or written. The platform can create define and undefine indices. **CLEAR (0):** NV indices that have TPMA_PLATFORM_CREATE SET may not be read or written (TPM_RC_HANDLE). The platform cannot define (TPM_RC_HIERARCHY) or undefined (TPM_RC_HANDLE) indices.
30:4	Reserved	shall be zero
31	orderly	**SET (1):** The TPM received a TPM2_Shutdown() and a matching TPM2_Startup(). **CLEAR (0):** TPM2_Startup(TPM_SU_CLEAR) was not preceded by a TPM2_Shutdown() of any type.

NOTE 1 Regarding phEnable, shEnable, ehEnable, and phEnableNV see ISO/IEC 11889-3, clause 25.2, "TPM2_HierarchyControl" for details on changing these attributes.

NOTE 2 Regarding phEnableNV, read refers to these commands: TPM2_NV_Read, TPM2_NV_ReadPublic, TPM_NV_Certify and TPM2_PolicyNV.

NOTE 3 Regarding phEnableNV, write refers to these commands: TPM2_NV_Write, TPM2_NV_Increment, TPM2_NV_Extend and TPM2_NV_SetBits.

NOTE 4 Regarding phEnableNV, the TPM needs to query the index TPMA_PLATFORM_CREATE attribute to determine whether phEnableNV is applicable. Since the TPM will return TPM_RC_HANDLE if the index does not exist, it also returns this error code if the index is disabled. Otherwise, the TPM would leak the existence of an index even when disabled.

NOTE 5 Regarding orderly, 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_State_Save(TPM_SU_STATE).

9.8 TPMA_MEMORY

This structure of this attribute is used to report the memory management method used by the TPM for transient objects and authorization sessions. This structure may be read using TPM2_GetCapability(*capability* = TPM_CAP_TPM_PROPERTIES, *property* = TPM_PT_MEMORY).

If the RAM memory is shared, then context save of a session may make it possible to load an additional transient object.

Table 36 — Definition of (UINT32) TPMA_MEMORY Bits <Out>

Bit	Name	Definition
0	sharedRAM	**SET (1):** indicates that the RAM memory used for authorization session contexts is shared with the memory used for transient objects **CLEAR (0):** indicates that the memory used for authorization sessions is not shared with memory used for transient objects
1	sharedNV	**SET (1):** indicates that the NV memory used for persistent objects is shared with the NV memory used for NV Index values **CLEAR (0):** indicates that the persistent objects and NV Index values are allocated from separate sections of NV
2	objectCopiedToRam	**SET (1):** indicates that the TPM copies persistent objects to a transient-object slot in RAM when the persistent object is referenced in a command. The TRM is required to make sure that an object slot is available. **CLEAR (0):** indicates that the TPM does not use transient-object slots when persistent objects are referenced
31:3	Reserved	shall be zero

9.9 TPMA_CC (Command Code Attributes)

9.9.1 Introduction

This structure defines the attributes of a command from a context management perspective. The fields of the structure indicate to the TPM Resource Manager (TRM) the number of resources required by a command and how the command affects the TPM's resources.

This structure is only used in a list returned by the TPM in response to TPM2_GetCapability(capability = TPM_CAP_COMMANDS).

For a command to the TPM, only the *commandIndex* field and *V* attribute are allowed to be non-zero.

9.9.2 Structure Definition

Table 37 — Definition of (TPM_CC) TPMA_CC Bits <OUT>

Bit	Name	Definition
15:0	commandIndex	indicates the command being selected
21:16	Reserved	shall be zero
22	nv	**SET (1):** indicates that the command may write to NV **CLEAR (0):** indicates that the command does not write to NV
23	extensive	**SET (1):** This command could flush any number of loaded contexts. **CLEAR (0):** no additional changes other than indicated by the *flushed* attribute
24	flushed	**SET (1):** The context associated with any transient handle in the command will be flushed when this command completes. **CLEAR (0):** No context is flushed as a side effect of this command.
27:25	cHandles	indicates the number of the handles in the handle area for this command
28	rHandle	**SET (1):** indicates the presence of the handle area in the response
29	V	**SET (1):** indicates that the command is vendor-specific **CLEAR (0):** indicates that the command is defined in ISO/IEC 11889
31:30	Res	allocated for software; shall be zero

9.9.3 Field Descriptions

9.9.3.1 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 ISO/IEC 11889. If *V* is one, then the meaning of *commandIndex* is as determined by the TPM vendor.

9.9.3.2 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.

9.9.3.3 Bit[23] – *extensive*

If this attribute is SET, then the TPM may flush many transient objects as a side effect of this command. In ISO/IEC 11889-3, a command that has this attribute is indicated by using a “{E}” decoration in the “Description” column of the *commandCode* parameter.

EXAMPLE See ISO/IEC 11889-3, clause 25.6, “TPM2_Clear”.

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

9.9.3.4 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 1 The TRM is expected to use this value to determine how many objects are loaded into transient TPM memory.

NOTE 2 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 behaviour of TPM2_ContextSave(), and sessions are flushed when context saved, but objects are not. The *flushed* attribute for that command shall be CLEAR.

In ISO/IEC 11889-3, a command that has this attribute is indicated by using a “{F}” decoration in the “Description” column of the *commandCode* parameter.

EXAMPLE See ISO/IEC 11889-3, clause 18.5, “TPM2_SequenceComplete”.

9.9.3.5 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).

9.9.3.6 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 session area.

9.9.3.7 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 ISO/IEC 11889.

9.9.3.8 Bits[31:30] – Res

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

10 Interface Types

10.1 Introduction

Clause 10 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.

10.2 TPMI_YES_NO

This interface type is used in place of a Boolean type in order to eliminate ambiguity in the handling of a octet that conveys a single bit of information. This type only has two allowed values, YES (1) and NO (0).

NOTE This list is not used as input to the TPM.

Table 38 — Definition of (BYTE) TPMI_YES_NO Type

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

10.3 TPMI_DH_OBJECT

The TPMI_DH_OBJECT interface type is a handle that references a loaded object. The handles in this set are used to refer to either transient or persistent object. The range of these values would change according to the TPM implementation.

NOTE These interface types not supposed to be used by system software to qualify the keys produced by the TPM. The value returned by the TPM will be used to reference the object.

Table 39 — 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	

10.4 TPMI_DH_PERSISTENT

The TPMI_DH_PERSISTENT interface type is a handle that references a location for a transient object. This type is used in TPM2_EvictControl() to indicate the handle to be assigned to the persistent object.

Table 40 — Definition of (TPM_HANDLE) TPMI_DH_PERSISTENT Type

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

10.5 TPMI_DH_ENTITY

The TPMI_DH_ENTITY interface type is TPM-defined values that are used to indicate that the handle refers to an *authValue*. The range of these values would change according to the TPM implementation.

Table 41 — Definition of (TPM_HANDLE) TPMI_DH_ENTITY Type <IN>

Values	Comments
TPM_RH_OWNER	
TPM_RH_ENDORSEMENT	
TPM_RH_PLATFORM	
TPM_RH_LOCKOUT	
{TRANSIENT_FIRST : TRANSIENT_LAST}	range of object handles
{PERSISTENT_FIRST : PERSISTENT_LAST}	
{NV_INDEX_FIRST : NV_INDEX_LAST}	
{PCR_FIRST : PCR_LAST}	
{TPM_RH_AUTH_00 : TPM_RH_AUTH_FF}	range of vendor-specific authorization values
+TPM_RH_NULL	conditional value
#TPM_RC_VALUE	

10.6 TPMI_DH_PCR

This interface type consists of the handles that may be used as PCR references. The upper end of this range of values would change according to the TPM implementation.

NOTE 1 Typically, the 0^{th} PCR will have a handle value of zero.

NOTE 2 The handle range for PCR is defined to be the same as the handle range for PCR in ISO/IEC 11889 (first edition).

Table 42 — Definition of (TPM_HANDLE) TPMI_DH_PCR Type <IN>

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

10.7 TPMI_SH_AUTH_SESSION

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

Table 43 — Definition of (TPM_HANDLE) TPMI_SH_AUTH_SESSION Type <IN/OUT>

Values	Comments
{HMAC_SESSION_FIRST : HMAC_SESSION_LAST}	range of HMAC authorization session handles
{POLICY_SESSION_FIRST: POLICY_SESSION_LAST}	range of policy authorization session handles
+TPM_RS_PW	a password authorization
#TPM_RC_VALUE	error returned if the handle is out of range

10.8 TPMI_SH_HMAC

This interface type is used for an authorization handle when the authorization session uses an HMAC.

Table 44 — Definition of (TPM_HANDLE) TPMI_SH_HMAC Type <IN/OUT>

Values	Comments
{HMAC_SESSION_FIRST: HMAC_SESSION_LAST}	range of HMAC authorization session handles
#TPM_RC_VALUE	error returned if the handle is out of range

10.9 TPMI_SH_POLICY

This interface type is used for a policy handle when it appears in a policy command.

Table 45 — Definition of (TPM_HANDLE) TPMI_SH_POLICY Type <IN/OUT>

Values	Comments
{POLICY_SESSION_FIRST: POLICY_SESSION_LAST}	range of policy authorization session handles
#TPM_RC_VALUE	error returned if the handle is out of range

10.10 TPMI_DH_CONTEXT

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

Table 46 — 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	

10.11 TPMI_RH_HIERARCHY

The TPMI_RH_HIERARCHY interface type is used as the type of a handle in a command when the handle is required to be one of the hierarchy selectors.

Table 47 — Definition of (TPM_HANDLE) TPMI_RH_HIERARCHY Type

Values	Comments
TPM_RH_OWNER	Storage hierarchy
TPM_RH_PLATFORM	Platform hierarchy
TPM_RH_ENDORSEMENT	Endorsement hierarchy
+TPM_RH_NULL	no hierarchy
#TPM_RC_VALUE	response code returned when the unmarshaling of this type fails

10.12 TPMI_RH_ENABLES

The TPMI_RH_ENABLES interface type is used as the type of a handle in a command when the handle is required to be one of the hierarchy or NV enables.

Table 48 — Definition of (TPM_HANDLE) TPMI_RH_ENABLES Type

Values	Comments
TPM_RH_OWNER	Storage hierarchy
TPM_RH_PLATFORM	Platform hierarchy
TPM_RH_ENDORSEMENT	Endorsement hierarchy
TPM_RH_PLATFORM_NV	Platform NV
+TPM_RH_NULL	no hierarchy
#TPM_RC_VALUE	response code returned when the unmarshaling of this type fails

10.13 TPMI_RH_HIERARCHY_AUTH

This interface type is used as the type of a handle in a command when the handle is required to be one of the hierarchy selectors or the Lockout Authorization.

Table 49 — Definition of (TPM_HANDLE) TPMI_RH_HIERARCHY_AUTH Type <IN>

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

10.14 TPMI_RH_PLATFORM

The TPMI_RH_PLATFORM interface type is used as the type of a handle in a command when the only allowed handle is TPM_RH_PLATFORM indicating that Platform Authorization is required.

Table 50 — Definition of (TPM_HANDLE) TPMI_RH_PLATFORM Type <IN>

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

10.15 TPMI_RH_OWNER

This interface type is used as the type of a handle in a command when the only allowed handle is TPM_RH_OWNER indicating that Owner Authorization is required.

Table 51 — Definition of (TPM_HANDLE) TPMI_RH_OWNER Type <IN>

Values	Comments
TPM_RH_OWNER	Owner hierarchy
+TPM_RH_NULL	may allow the null handle
#TPM_RC_VALUE	response code returned when the unmarshaling of this type fails

10.16 TPMI_RH_ENDORSEMENT

This interface type is used as the type of a handle in a command when the only allowed handle is TPM_RH_ENDORSEMENT indicating that Endorsement Authorization is required.

Table 52 — Definition of (TPM_HANDLE) TPMI_RH_ENDORSEMENT Type <IN>

Values	Comments
TPM_RH_ENDORSEMENT	Endorsement hierarchy
+TPM_RH_NULL	may allow the null handle
#TPM_RC_VALUE	response code returned when the unmarshaling of this type fails

10.17 TPMI_RH_PROVISION

The TPMI_RH_PROVISION interface type is used as the type of the handle in a command when the only allowed handles are either TPM_RH_OWNER or TPM_RH_PLATFORM indicating that either Platform Authorization or Owner Authorization are allowed.

In most cases, either Platform Authorization or Owner Authorization may be used to authorize the commands used for management of the resources of the TPM and this interface type will be used.

Table 53 — Definition of (TPM_HANDLE) TPMI_RH_PROVISION Type <IN>

Value	Comments
TPM_RH_OWNER	handle for Owner Authorization
TPM_RH_PLATFORM	handle for Platform Authorization
#TPM_RC_VALUE	response code returned when the unmarshaling of this type fails

10.18 TPMI_RH_CLEAR

The TPMI_RH_CLEAR interface type is used as the type of the handle in a command when the only allowed handles are either TPM_RH_LOCKOUT or TPM_RH_PLATFORM indicating that either Platform Authorization or Lockout Authorization are allowed.

This interface type is normally used for performing or controlling TPM2_Clear().

Table 54 — Definition of (TPM_HANDLE) TPMI_RH_CLEAR Type <IN>

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

10.19 TPMI_RH_NV_AUTH

This interface type is used to identify the source of the authorization for access to an NV location. The handle value of a TPMI_RH_NV_AUTH shall indicate that the authorization value is either Platform Authorization, Owner Authorization, or the *authValue.* This type is used in the commands that access an NV Index (commands of the form TPM2_NV_xxx) other than TPM2_NV_DefineSpace() and TPM2_NV_UndefineSpace().

Table 55 — Definition of (TPM_HANDLE) TPMI_RH_NV_AUTH Type <IN>

Value	Comments
TPM_RH_PLATFORM	Platform Authorization is allowed
TPM_RH_OWNER	Owner Authorization is allowed
{NV_INDEX_FIRST:NV_INDEX_LAST}	range for NV locations
#TPM_RC_VALUE	response code returned when unmarshaling of this type fails

10.20 TPMI_RH_LOCKOUT

The TPMI_RH_LOCKOUT interface type is used as the type of a handle in a command when the only allowed handle is TPM_RH_LOCKOUT indicating that Lockout Authorization is required.

Table 56 — Definition of (TPM_HANDLE) TPMI_RH_LOCKOUT Type <IN>

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

10.21 TPMI_RH_NV_INDEX

This interface type is used to identify an NV location. This type is used in the NV commands.

Table 57 — Definition of (TPM_HANDLE) TPMI_RH_NV_INDEX Type <IN/OUT>

Value	Comments
{NV_INDEX_FIRST:NV_INDEX_LAST}	Range of NV Indexes
#TPM_RC_VALUE	error returned if the handle is out of range

10.22 TPMI_ALG_HASH

A TPMI_ALG_HASH is an interface type of all the hash algorithms implemented on a specific TPM. Table 58 is a list 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 An implementation would modify this table according to the implemented algorithms, changing the values that are accepted as hash algorithms.

Table 58 — Definition of (TPM_ALG_ID) TPMI_ALG_HASH Type

Values	Comments
TPM_ALG_SHA1	example
TPM_ALG_SHA256	example
TPM_ALG_SM3_256	example
TPM_ALG_SHA384	example
TPM_ALG_SHA512	example
+TPM_ALG_NULL	
#TPM_RC_HASH	

10.23 TPMI_ALG_ASYM (Asymmetric Algorithms)

A TPMI_ALG_ASYM is an interface type of all the asymmetric algorithms implemented on a specific TPM. Table 59 lists each of the asymmetric algorithms that have an algorithm ID assigned by the TCG.

Table 59 — Definition of (TPM_ALG_ID) TPMI_ALG_ASYM Type

Values	Comments
TPM_ALG_RSA	
TPM_ALG_ECC	
+TPM_ALG_NULL	
#TPM_RC_ASYMMETRIC	

10.24 TPMI_ALG_SYM (Symmetric Algorithms)

A TPMI_ALG_SYM is an interface type of all the symmetric algorithms that have an algorithm ID assigned by the TCG and are implemented on the TPM.

The list in the table below is illustrative and will change according to the implementation. The validation code will only accept the subset of algorithms implemented on a 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 60 — Definition of (TPM_ALG_ID) TPMI_ALG_SYM Type

Values	Comments
TPM_ALG_AES	example
TPM_ALG_SM4	example
TPM_ALG_CAMELLIA	example
TPM_ALG_XOR	example
+TPM_ALG_NULL	required to be present in all versions of this table
#TPM_RC_SYMMETRIC	

10.25 TPMI_ALG_SYM_OBJECT

A TPMI_ALG_SYM_OBJECT is an interface type of all the TCG-defined symmetric algorithms that may be used as companion symmetric encryption algorithm for an asymmetric object. All algorithms in this list shall be block ciphers usable in Cipher Feedback (CFB).

Table 61 is illustrative. It would be modified to indicate the algorithms of the TPM.

NOTE TPM_ALG_XOR cannot be in this list.

Table 61 — Definition of (TPM_ALG_ID) TPMI_ALG_SYM_OBJECT Type

Values	Comments
TPM_ALG_AES	example
TPM_ALG_SM4	example
TPM_ALG_CAMELLIA	example
+TPM_ALG_NULL	required to be present in all versions of this table
#TPM_RC_SYMMETRIC	

10.26 TPMI_ALG_SYM_MODE

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

This version of the table is not expected to be the table checked by the validation code. Rather, the table would be replaced by one containing the algorithms implemented on the TPM and the values in that table would be checked by the input validation code.

Table 62 — Definition of (TPM_ALG_ID) TPMI_ALG_SYM_MODE Type

Values	Comments
TPM_ALG_CTR	IV will be determined by use. If the outside provides the nonce and initial counter, then the caller can know what IV to provide for chaining.
TPM_ALG_OFB	XOR last cipher text block with last plaintext to create IV for next block
TPM_ALG_CBC	IV will be determined by use. indefinite chaining using previous output block as IV for next block
TPM_ALG_CFB	shall be implemented in all TPM compliant with ISO/IEC 11889 IV will be determined by use. indefinite chaining using previous cipher text as IV
TPM_ALG_ECB	no IV or chaining value required
+TPM_ALG_NULL	
#TPM_RC_MODE	

Implementation of TPM_ALG_CFB is mandatory. CFB is specified ISO/IEC 10116:2006, making ISO/IEC 10116:2006 indispensable for an implementation of this International Standard.

10.27 TPMI_ALG_KDF (Key and Mask Generation Functions)

A TPMI_ALG_KDF is an interface type of all the key derivation functions implemented on a specific TPM. Table 63 is exemplary and would change based on the algorithms implemented in a TPM.

Table 63 — Definition of (TPM_ALG_ID) TPMI_ALG_KDF Type

Values	Comments
TPM_ALG_MGF1	
TPM_ALG_KDF1_SP800_108	
TPM_ALG_KDF1_SP800_56a	
TPM_ALG_KDF2	
+TPM_ALG_NULL	
#TPM_RC_KDF	

10.28 TPMI_ALG_SIG_SCHEME

This is the definition of the interface type for a signature scheme. This table would change according to the algorithms implemented on the TPM.

Table 64 — Definition of (TPM_ALG_ID) TPMI_ALG_SIG_SCHEME Type

Values	Comments
TPM_ALG_RSASSA	requires that RSA be implemented
TPM_ALG_RSAPSS	requires that RSA be implemented
TPM_ALG_ECDSA	requires that ECC be implemented
TPM_ALG_ECDAA	requires that ECC and ECDAA be implemented
TPM_ALG_ECSCHNORR	
TPM_ALG_SM2	requires that ECC be implemented
TPM_ALG_HMAC	present on all TPM
+TPM_ALG_NULL	
#TPM_RC_SCHEME	response code when a signature scheme is not correct

Implementation of TPM_ALG_HMAC is mandatory. HMAC is specified ISO/IEC 9797-2, making ISO/IEC 9797-2 indispensable for an implementation of this International Standard.

10.29 TPMI_ECC_KEY_EXCHANGE

This is the definition of the interface type for an ECC key exchange scheme. This table would change according to the algorithms implemented on the TPM.

Table 65 — Definition of (TPM_ALG_ID) TPMI_ECC_KEY_EXCHANGE Type

Values	Comments
TPM_ALG_ECDH	used for single and two phase key exchange
TPM_ALG_ECMQV	
TPM_ALG_SM2	requires that ECC be implemented
+TPM_ALG_NULL	
#TPM_RC_SCHEME	response code when a key exchange scheme is not correct

10.30 TPMI_ST_COMMAND_TAG

This interface type is used for the command tags.

The response code for a bad command tag has the same value as the ISO/IEC 11889 (first edition) response code (TPM_BAD_TAG). This value is used in case the software is not compatible with ISO/IEC 11889 and an unexpected response code might have unexpected side effects.

Table 66 — Definition of (TPM_ST) TPMI_ST_COMMAND_TAG Type

Values	Comments
TPM_ST_NO_SESSIONS	
TPM_ST_SESSIONS	

Values	Comments
#TPM_RC_BAD_TAG	

11 Structure Definitions

11.1 TPMS_EMPTY

This structure is used as a placeholder. In some cases, a union will have a selector value with no data to unmarshal when that type is selected. Rather than leave the entry empty, TPMS_EMPTY may be selected. Alternatively, a more descriptive value may be created as a type of TPMS_EMPTY (such as, TPMS_SCHEME_RSAES).

NOTE The tool chain will special case this structure and create the marshaling and unmarshaling code for this structure but not create a type definition. The unmarshaling code for this structure will return TPM_RC_SUCCESS and the marshaling code will return 0.

Table 67 — Definition of TPMS_EMPTY Structure <IN/OUT>

Parameter	Type	Description
		a structure with no member

11.2 TPMS_ALGORITHM_DESCRIPTION

This structure is a return value for a TPM2_GetCapability() that reads the installed algorithms.

Table 68 — Definition of TPMS_ALGORITHM_DESCRIPTION Structure <OUT>

Parameter	Type	Description
alg	TPM_ALG_ID	an algorithm
attributes	TPMA_ALGORITHM	the attributes of the algorithm

11.3 Hash/Digest Structures

11.3.1 TPMU_HA (Hash)

A TPMU_HA is a union of all the hash algorithms implemented on a TPM. Table 69 is exemplary and would change based on the algorithms implemented in a TPM.

NOTE If processed by an automated tool, each entry of the table ought to be qualified (with #ifdef/#endif) so that if the hash algorithm is not implemented on the TPM, the parameter associated with that hash is not present. This will keep the union from being larger than the largest digest of a hash implemented on that TPM.

Table 69 — Definition of TPMU_HA Union <IN/OUT, S>

Parameter	Type	Selector	Description
sha1 [SHA1_DIGEST_SIZE]	BYTE	TPM_ALG_SHA1	
sha256 [SHA256_DIGEST_SIZE]	BYTE	TPM_ALG_SHA256	
sm3_256 [SM3_256_DIGEST_SIZE]	BYTE	TPM_ALG_SM3_256	
sha384 [SHA384_DIGEST_SIZE]	BYTE	TPM_ALG_SHA384	
sha512 [SHA512_DIGEST_SIZE]	BYTE	TPM_ALG_SHA512	
null		TPM_ALG_NULL	

11.3.2 TPMT_HA

Table 70 shows the basic hash-agile structure used in ISO/IEC 11889. 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 70 — Definition of TPMT_HA Structure <IN/OUT>

Parameter	Type	Description
hashAlg	+TPMI_ALG_HASH	selector of the hash contained in the *digest* that implies the size of the *digest*
[hashAlg] digest	TPMU_HA	the digest data
NOTE	The leading “+” on the type indicates that this structure ought to 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.	

11.4 Sized Buffers

11.4.1 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.

When the type of the second parameter in a TPM2B_ structure is BYTE, the TPM shall unmarshal the indicated number of octets, which may be zero.

When the type of the second parameter in the TPM2B_ structure is not BYTE, the value of the *size* field shall either be zero indicating that no structure is to be unmarshaled; or it shall be identical to the number of octets unmarshaled for the second parameter.

NOTE 1 If the TPM2B_ defines a structure and not an array of octets, then the structure is self-describing and the TPM will be able to determine how many octets are in the structure when it is unmarshaled. If that number of octets is not equal to the size parameter, then it is an error.

NOTE 2 The reason that a structure can be put into a TPM2B_ is that the parts of the structure can 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 facilitates the structure to be manipulated as an opaque block. Placing a structure in a TPM2B_ also makes it possible to use parameter encryption on the structure.

If a TPM2B_ is encrypted, the TPM will encrypt/decrypt the data field of the TPM2B_ but not the *size* parameter. The TPM will encrypt/decrypt the number of octets indicated by the *size* field.

NOTE 3 In the reference implementation, a TPM2B type is defined that is a 16-bit size field followed by a single byte of data. The TPM2B_ is then defined as a union that contains a TPM2B (union member ‘b’) and the structure in the definition table (union member ‘t’). This union is used for internally generated structures so that there is a way to define a structure of the correct size (forced by the ‘t’ member) while giving a way to pass the structure generically as a ‘b’. Most function calls use the 't' member so that the compiler will generate a warning if there is a type error (a TPM2B_ of the wrong type). Having the type checked helps avoid many issues with buffer overflow caused by a too small buffer being passed to a function.

11.4.2 TPM2B_DIGEST

This structure is used for a sized buffer that cannot be larger than the largest digest produced by any hash algorithm implemented on the TPM.

As with all sized buffers, the size is checked to see if it is within the prescribed range. If not, the response code is TPM_RC_SIZE.

NOTE For any structure, like the one below, that contains an implied size check, it is implied that TPM_RC_SIZE is a possible response code and the response code will not be listed in the table.

Table 71 — Definition of TPM2B_DIGEST Structure

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

11.4.3 TPM2B_DATA

This structure is used for a data buffer that is required to be no larger than the size of the Name of an object. This size limit includes the algorithm ID of the hash and the hash data.

Table 72 — Definition of TPM2B_DATA Structure

Parameter	Type	Description
size	UINT16	size in octets of the *buffer* field; may be 0
buffer[size]{:sizeof(TPMT_HA)}	BYTE	the buffer area that contains the algorithm ID and the digest

11.4.4 TPM2B_NONCE

Table 73 — Definition of Types for TPM2B_NONCE

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

11.4.5 TPM2B_AUTH

This structure is used for an authorization value and limits an *authValue* to being no larger than the largest digest produced by a TPM. In order to ensure consistency within an object, the *authValue* may be no larger than the size of the digest produced by the object's *nameAlg*. This ensures that any TPM that can load the object will be able to handle the *authValue* of the object.

Table 74 — Definition of Types for TPM2B_AUTH

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

11.4.6 TPM2B_OPERAND

This type is a sized buffer that can hold an operand for a comparison with an NV Index location. The maximum size of the operand is implementation dependent but a TPM is required to support an operand size that is at least as big as the digest produced by any of the hash algorithms implemented on the TPM.

Table 75 — Definition of Types for TPM2B_OPERAND

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

11.4.7 TPM2B_EVENT

This type is a sized buffer that can hold event data.

Table 76 — Definition of TPM2B_EVENT Structure

Parameter	Type	Description
size	UINT16	size of the operand *buffer*
buffer [size] {:1024}	BYTE	the operand

11.4.8 TPM2B_MAX_BUFFER

This type is a sized buffer that can hold a maximally sized buffer for commands that use a large data buffer.

EXAMPLE Examples of commands that might use large data buffers are TPM2_PCR_Event(), TPM2_Hash(), TPM2_SequenceUpdate(), or TPM2_FieldUpgradeData().

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

Table 77 — Definition of TPM2B_MAX_BUFFER Structure

Parameter	Type	Description
size	UINT16	size of the buffer
buffer [size] {:MAX_DIGEST_BUFFER}	BYTE	the operand
NOTE MAX_DIGEST_BUFFER is TPM-dependent but is required to be at least 1,024.		

11.4.9 TPM2B_MAX_NV_BUFFER

This type is a sized buffer that can hold a maximally sized buffer for NV data commands.

EXAMPLE Examples of NV data commands are TPM2_NV_Read(), TPM2_NV_Write(), and TPM2_NV_Certify().

Table 78 — Definition of TPM2B_MAX_NV_BUFFER Structure

Parameter	Type	Description
size	UINT16	size of the buffer
buffer [size] {:MAX_NV_BUFFER_SIZE}	BYTE	the operand
NOTE MAX_NV_BUFFER_SIZE is TPM-dependent.		

11.4.10 TPM2B_TIMEOUT

This TPM-dependent structure is used to provide the timeout value for an authorization.

Table 79 — Definition of TPM2B_TIMEOUT Structure <IN/OUT>

Parameter	Type	Description
size	UINT16	size of the timeout value This value is fixed for a TPM implementation.
buffer [size] {:sizeof(UINT64)}	BYTE	the timeout value

11.4.11 TPM2B_IV

This structure is used for passing an initial value for a symmetric block cipher to or from the TPM. The size is set to be the largest block size of any implemented symmetric cipher implemented on the TPM.

Table 80 — Definition of TPM2B_IV Structure <IN/OUT>

Parameter	Type	Description
size	UINT16	size of the timeout value This value is fixed for a TPM implementation.
buffer [size] {:MAX_SYM_BLOCK_SIZE}	BYTE	the timeout value

11.5 Names

11.5.1 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.

11.5.2 TPMU_NAME

Table 81 — 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

11.5.3 TPM2B_NAME

This buffer holds a Name for any entity type.

The type of Name in the structure is determined by context and the *size* parameter. If *size* is four, then the Name is a handle. If *size* is zero, then no Name is present. Otherwise, the size shall be the size of a TPM_ALG_ID plus the size of the digest produced by the indicated hash algorithm.

Table 82 — Definition of TPM2B_NAME Structure

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

11.6 PCR Structures

11.6.1 TPMS_PCR_SELECT

This structure provides a standard method of specifying a list of PCR.

PCR numbering starts at zero.

pcrSelect is an array of octets. The octet containing the bit corresponding to a specific PCR is found by dividing the PCR number by 8.

EXAMPLE 1 The bit in *pcrSelect* corresponding to PCR 19 is in *pcrSelect* [2] (19/8 = 2).

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

EXAMPLE 2 The bit in *pcrSelect* [2] corresponding to PCR 19 is bit 3 (19 mod 8). If *sizeofSelect* is 3, then the *pcrSelect* array that would specify PCR 19 and no other PCR is 00 00 08_{16}.

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 values 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 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 the number of PCR implemented on the TPM

If the TPM implements more PCR than there are bits in *pcrSelect*, the additional PCR are not selected.

EXAMPLE 3 If the applicable platform-specific specification requires that the TPM have a minimum of 24 PCR but the TPM implements 32, then a PCR select of 3 octets would imply that PCR 24-31 are not selected.

Table 83 — Definition of TPMS_PCR_SELECT Structure

Parameter	Type	Description
sizeofSelect {PCR_SELECT_MIN:}	UINT8	the size in octets of the *pcrSelect* array
pcrSelect [sizeofSelect] {:PCR_SELECT_MAX}	BYTE	the bit map of selected PCR
#TPM_RC_VALUE		

11.6.2 TPMS_PCR_SELECTION

Table 84 — Definition of TPMS_PCR_SELECTION Structure

Parameter	Type	Description
hash	TPMI_ALG_HASH	the hash algorithm associated with the selection
sizeofSelect {PCR_SELECT_MIN:}	UINT8	the size in octets of the *pcrSelect* array
pcrSelect [sizeofSelect] {:PCR_SELECT_MAX}	BYTE	the bit map of selected PCR
#TPM_RC_VALUE		

11.7 Tickets

11.7.1 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 11.7 are to be used by a TPM that is compliant with ISO/IEC 11889.

The method of creating the ticket data is:

$$\mathbf{HMAC}_{contexAlg}(proof, (ticketType \,||\, param \,\{\,||\, param\,\{...\}\}) \quad (3)$$

where

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

The proof value used for each hierarchy is shown in Table 85.

Table 85 — Values for *proof* Used in Tickets

Hierarchy	proof	Description
None	Empty Buffer	
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 86. This is a template for the tickets shown in the remainder of clause 11.7.

Table 86 — General Format of a Ticket

Parameter	Type	Description
tag	TPM_ST	structure tag indicating the type of the ticket
hierarchy	TPMI_RH_HIERARCHY+	the hierarchy of the proof value
digest	TPM2B_DIGEST	the HMAC over the ticket-specific data

11.7.2 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 needs for 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.

11.7.3 TPMT_TK_CREATION

This ticket is produced by TPM2_Create() or TPM2_CreatePrimary(). It is used to bind the creation data to the object to which it applies. The ticket is computed by

$$\mathbf{HMAC}_{contextAlg}(proof, (\text{TPM\_ST\_CREATION} \,||\, name \,||\, \mathbf{H}_{nameAlg}(\text{TPMS\_CREATION\_DATA}))) \quad (4)$$

where

$\mathbf{HMAC}_{contextAlg}()$	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
$\mathbf{H}_{nameAlg}()$	hash using the *nameAlg* of the created object
TPMS_CREATION_DATA	the creation data structure associated with name

Table 87 — Definition of TPMT_TK_CREATION Structure

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

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

11.7.4 TPMT_TK_VERIFIED

This ticket is produced by TPM2_VerifySignature(). This formulation is used for multiple ticket uses. The ticket provides evidence that the TPM has validated that a digest was signed by a key with the Name of keyName. The ticket is computed by

$$\mathbf{HMAC}_{contextAlg}(proof, (\text{TPM\_ST\_VERIFIED} \,||\, digest \,||\, keyName)) \quad (5)$$

where

$\mathbf{HMAC}_{contextAlg}()$	an HMAC using the context integrity hash
proof	a TPM secret value associated with the hierarchy associated with *keyName*
TPM_ST_VERIFIED	a value used to ensure that the ticket is properly used
digest	the signed digest
keyName	Name of the key that signed digest

Table 88 — Definition of TPMT_TK_VERIFIED Structure

Parameter	Type	Description
tag {TPM_ST_VERIFIED}	TPM_ST	ticket structure tag
#TPM_RC_TAG		error returned when *tag* is not TPM_ST_VERIFIED
hierarchy	TPMI_RH_HIERARCHY+	the hierarchy containing *keyName*
digest	TPM2B_DIGEST	This shall be the HMAC produced using a proof value of *hierarchy*.

EXAMPLE A NULL Verified Ticket is the tuple <TPM_ST_VERIFIED, TPM_RH_NULL, 0x0000>.

11.7.5 TPMT_TK_AUTH

This ticket is produced by TPM2_PolicySigned() and TPM2_PolicySecret() when the authorization has an expiration time. The ticket is computed by

$$\mathbf{HMAC}_{contextAlg}(proof, (\text{TPM\_ST\_AUTH\_xxx} \,||\, timeout \,||\, cpHash \,||\, policyRef \,||\, authName)) \quad (6)$$

where

$\mathbf{HMAC}_{contextAlg}()$	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
$timeout$	implementation-specific value indicating when the authorization expires
$cpHash$	optional hash of the authorized command
$policyRef$	optional reference to a policy value
$authName$	Name of the object that signed the authorization

Table 89 — Definition of TPMT_TK_AUTH Structure

Parameter	Type	Description
tag {TPM_ST_AUTH_SIGNED, TPM_ST_AUTH_SECRET}	TPM_ST	ticket structure tag
#TPM_RC_TAG		error returned when *tag* is not TPM_ST_AUTH
hierarchy	TPMI_RH_HIERARCHY+	the hierarchy of the object used to produce the ticket
digest	TPM2B_DIGEST	This shall be the HMAC produced using a proof value of *hierarchy*.

EXAMPLE A NULL Auth Ticket is the tuple <TPM_ST_AUTH_SIGNED, TPM_RH_NULL, 0x0000> or the tuple <TPM_ST_AUTH_SIGNED, TPM_RH_NULL, 0x0000>

11.7.6 TPMT_TK_HASHCHECK

This ticket is produced by TPM2_SequenceComplete() when the message that was digested did not start with TPM_GENERATED_VALUE. The ticket is computed by

$$\mathbf{HMAC}_{contexAlg}(proof, (\text{TPM\_ST\_HASHCHECK} \,||\, digest)) \quad (7)$$

where

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

Table 90 — Definition of TPMT_TK_HASHCHECK Structure

Parameter	Type	Description
tag {TPM_ST_HASHCHECK}	TPM_ST	ticket structure tag
#TPM_RC_TAG		error returned when is not TPM_ST_HASHCHECK
hierarchy	TPMI_RH_HIERARCHY+	the hierarchy
digest	TPM2B_DIGEST	This shall be the HMAC produced using a proof value of *hierarchy*.

11.8 Property Structures

11.8.1 TPMS_ALG_PROPERTY

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

Table 91 — Definition of TPMS_ALG_PROPERTY Structure <OUT>

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

11.8.2 TPMS_TAGGED_PROPERTY

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

Table 92 — Definition of TPMS_TAGGED_PROPERTY Structure <OUT>

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

11.8.3 TPMS_TAGGED_PCR_SELECT

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

Table 93 — Definition of TPMS_TAGGED_PCR_SELECT Structure <OUT>

Parameter	Type	Description
tag	TPM_PT	the property identifier
sizeofSelect {PCR_SELECT_MIN:}	UINT8	the size in octets of the *pcrSelect* array
pcrSelect [sizeofSelect] {:PCR_SELECT_MAX}	BYTE	the bit map of PCR with the identified property

11.9 Lists

11.9.1 TPML_CC

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

Table 94 — Definition of TPML_CC Structure

Parameter	Type	Description
count	UINT32	number of commands in the *commandCode* list; may be 0
commandCodes[count]{:MAX_CAP_CC}	TPM_CC	a list of command codes The maximum only applies to a command code list in a command. The response size is limited only by the size of the parameter buffer.
#TPM_RC_SIZE		response code when count is greater than the maximum allowed list size

11.9.2 TPML_CCA

This list is only used in TPM2_GetCapability(capability = TPM_CAP_COMMANDS).

The values in the list are returned in *commandIndex* order with vendor-specific commands returned after other commands. Because of the other attributes, the commands may not be returned in strict numerical order. They will be in *commandIndex* order.

Table 95 — Definition of TPML_CCA Structure <OUT>

Parameter	Type	Description
count	UINT32	number of values in the *commandAttributes* list; may be 0
commandAttributes[count]{:MAX_CAP_CC}	TPMA_CC	a list of command codes attributes

11.9.3 TPML_ALG

This list is returned by TPM2_IncrementalSelfTest().

Table 96 — Definition of TPML_ALG Structure

Parameter	Type	Description
count	UINT32	number of algorithms in the *algorithms* list; may be 0
algorithms[count]{:MAX_ALG_LIST_SIZE}	TPM_ALG_ID	a list of algorithm IDs The maximum only applies to an algorithm list in a command. The response size is limited only by the size of the parameter buffer.
#TPM_RC_SIZE		response code when *count* is greater than the maximum allowed list size

11.9.4 TPML_HANDLE

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

NOTE This list is not used as input to the TPM.

Table 97 — Definition of TPML_HANDLE Structure <OUT>

Name	Type	Description
count	UINT32	the number of handles in the list may have a value of 0
handle[count]{: MAX_CAP_HANDLES}	TPM_HANDLE	an array of handles
#TPM_RC_SIZE		response code when *count* is greater than the maximum allowed list size

11.9.5 TPML_DIGEST

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

Table 98 — Definition of TPML_DIGEST Structure

Parameter	Type	Description
count {2:}	UINT32	number of digests in the list, minimum is two for TPM2_PolicyOR().
digests[count]{:8}	TPM2B_DIGEST	a list of digests For TPM2_PolicyOR(), all digests will have been computed using the digest of the policy session. For TPM2_PCR_Read(), each digest will be the size of the digest for the bank containing the PCR.
#TPM_RC_SIZE		response code when *count* is not at least two or is greater than eight

11.9.6 TPML_DIGEST_VALUES

This list is used to convey a list of digest values. This type is returned by TPM2_Event() and TPM2_SequenceComplete() and is an input for TPM2_PCR_Extend().

NOTE 1 This construct limits the number of hashes in the list to the number of digests implemented in the TPM rather than the number of PCR banks. This allows extra values to appear in a call to TPM2_PCR_Extend().

NOTE 2 The digest for an unimplemented hash algorithm might not be in a list because the TPM might not recognize the algorithm as being a hash and it may not know the digest size.

Table 99 — Definition of TPML_DIGEST_VALUES Structure

Parameter	Type	Description
count	UINT32	number of digests in the list
digests[count]{:HASH_COUNT}	TPMT_HA	a list of tagged digests
#TPM_RC_SIZE		response code when *count* is greater than the possible number of banks

11.9.7 TPM2B_DIGEST_VALUES

Digest list in a sized buffer. This list is returned by TPM2_PCR_SequenceComplete().

Table 100 — Definition of TPM2B_DIGEST_VALUES Structure

Parameter	Type	Description
size	UINT16	size of the operand *buffer*
buffer [size] {:sizeof(TPML_DIGEST_VALUES)}	BYTE	the operand

11.9.8 TPML_PCR_SELECTION

This list is used to indicate the PCR that are included in a selection when more than one PCR value may be selected.

This structure is an input parameter to TPM2_PolicyPCR() to indicate the PCR that will be included in the digest of PCR for the authorization. The structure is used in TPM2_PCR_Read() command to indicate the PCR values to be returned and in the response to indicate which PCR are included in the list of returned digests. The structure is an output parameter from TPM2_Create() and indicates the PCR used in the digest of the PCR state when the object was created. The structure is also contained in the attestation structure of TPM2_Quote().

When this structure is used to select PCR to be included in a digest, the selected PCR are concatenated to create a “message” containing all of the PCR, and then the message is hashed using the context-specific hash algorithm.

Table 101 — Definition of TPML_PCR_SELECTION Structure

Parameter	Type	Description
count	UINT32	number of selection structures A value of zero is allowed.
pcrSelections[count]{:HASH_COUNT}	TPMS_PCR_SELECTION	list of selections
#TPM_RC_SIZE		response code when *count* is greater than the possible number of banks

11.9.9 TPML_ALG_PROPERTY

This list is used to report on a list of algorithm attributes. It is returned in a TPM2_GetCapability().

Table 102 — Definition of TPML_ALG_PROPERTY Structure <OUT>

Parameter	Type	Description
count	UINT32	number of algorithm properties structures A value of zero is allowed.
algProperties[count]{:MAX_CAP_ALGS}	TPMS_ALG_PROPERTY	list of properties

11.9.10 TPML_TAGGED_TPM_PROPERTY

This list is used to report on a list of properties that are TPMS_TAGGED_PROPERTY values. It is returned by a TPM2_GetCapability().

Table 103 — Definition of TPML_TAGGED_TPM_PROPERTY Structure <OUT>

Parameter	Type	Description
count	UINT32	number of properties A value of zero is allowed.
tpmProperty[count]{:MAX_TPM_PROPERTIES}	TPMS_TAGGED_PROPERTY	an array of tagged properties

11.9.11 TPML_TAGGED_PCR_PROPERTY

This list is used to report on a list of properties that are TPMS_PCR_SELECT values. It is returned by a TPM2_GetCapability().

Table 104 — Definition of TPML_TAGGED_PCR_PROPERTY Structure <OUT>

Parameter	Type	Description
count	UINT32	number of properties A value of zero is allowed.
pcrProperty[count]{:MAX_PCR_PROPERTIES}	TPMS_TAGGED_PCR_SELECT	a tagged PCR selection

11.9.12 TPML_ECC_CURVE

This list is used to report the ECC curve ID values supported by the TPM. It is returned by a TPM2_GetCapability().

Table 105 — Definition of {ECC} TPML_ECC_CURVE Structure <OUT>

Parameter	Type	Description
count	UINT32	number of curves A value of zero is allowed.
eccCurves[count]{:MAX_ECC_CURVES}	TPM_ECC_CURVE	array of ECC curve identifiers

11.10 Capabilities Structures

11.10.1 TPMU_CAPABILITIES

Table 106 — Definition of TPMU_CAPABILITIES Union <OUT>

Parameter	Type	Selector	Description
algorithms	TPML_ALG_PROPERTY	TPM_CAP_ALGS	
handles	TPML_HANDLE	TPM_CAP_HANDLES	
command	TPML_CCA	TPM_CAP_COMMANDS	
ppCommands	TPML_CC	TPM_CAP_PP_COMMANDS	
auditCommands	TPML_CC	TPM_CAP_AUDIT_COMMANDS	
assignedPCR	TPML_PCR_SELECTION	TPM_CAP_PCRS	
tpmProperties	TPML_TAGGED_TPM_PROPERTY	TPM_CAP_TPM_PROPERTIES	
pcrProperties	TPML_TAGGED_PCR_PROPERTY	TPM_CAP_PCR_PROPERTIES	
eccCurves	TPML_ECC_CURVE	TPM_CAP_ECC_CURVES	TPM_ALG_ECC

11.10.2 TPMS_CAPABILITY_DATA

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

Table 107 — Definition of TPMS_CAPABILITY_DATA Structure <OUT>

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

11.11 Clock/Counter Structures

11.11.1 TPMS_CLOCK_INFO

This structure is used in each of the attestation commands.

Table 108 — Definition of TPMS_CLOCK_INFO Structure

Parameter	Type	Description
clock	UINT64	time in milliseconds during which the TPM has been powered This structure element is used to report on the TPM's *Clock* value. The value of *Clock* shall be recorded in non-volatile memory no less often than once per 2^{22} milliseconds (~69.9 minutes) of TPM operation. The reference for the millisecond timer is the TPM oscillator. This value is reset to zero when the Storage Primary Seed is changed (TPM2_Clear()). This value may be advanced by TPM2_AdvanceClock().
resetCount	UINT32	number of occurrences of TPM Reset since the last TPM2_Clear()
restartCount	UINT32	number of times that TPM2_Shutdown() or _TPM_Hash_Start have occurred since the last TPM Reset or TPM2_Clear().
safe	TPMI_YES_NO	no value of *Clock* greater than the current value of *Clock* has been previously reported by the TPM. Set to YES on TPM2_Clear().

11.11.2 *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. 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() with *capability* = TPM_CAP_TPM_PROPERTIES and *property* = TPM_PT_CLOCK_UPDATE.

11.11.3 *ResetCount*

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

11.11.4 *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().

11.11.5 *Safe*

This parameter is set to YES when the value reported in *Clock* is guaranteed to be unique for the current Owner. It is set to NO when the value of *Clock* may have been reported in a previous attestation or access.

This parameter will be YES if a TPM2_Startup() was preceded by TPM2_Shutdown() with no intervening commands. It will also be YES after an update of 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().

11.11.6 TPMS_TIME_INFO

This structure is used in the TPM2_TICK attestation.

The *Time* value reported in this structure is reset whenever the TPM is reset. An implementation may reset the value of *Time* any time after _TPM_Init and before the TPM returns after TPM2_Startup(). The value of *Time* shall increment continuously while power is applied to the TPM.

Table 109 — Definition of TPMS_TIME_INFO Structure

Parameter	Type	Description
time	UINT64	time in milliseconds since the last _TPM_Init() or TPM2_Startup() This structure element is used to report on the TPM's *Time* value.
clockInfo	TPMS_CLOCK_INFO	a structure containing the clock information

11.12 TPM Attestation Structures

11.12.1 Introduction

Clause 11.12 describes the structures that are used when a TPM creates a structure to be signed. The signing structures follow a standard format TPM2B_ATTEST with case-specific information embedded.

11.12.2 TPMS_TIME_ATTEST_INFO

This structure is used when the TPM performs TPM2_GetClock.

Table 110 — Definition of TPMS_TIME_ATTEST_INFO Structure <OUT>

Parameter	Type	Description
time	TPMS_TIME_INFO	the *Time*, *clock*, *resetCount*, *restartCount*, and *Safe* indicator
firmwareVersion	UINT64	a vendor-specific value indicating the version number of the firmware

11.12.3 TPMS_CERTIFY_INFO

This is the attested data for TPM2_Certify().

Table 111 — Definition of TPMS_CERTIFY_INFO Structure <OUT>

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

11.12.1 TPMS_QUOTE_INFO

This is the *attested* data for TPM2_Quote().

Table 112 — Definition of TPMS_QUOTE_INFO Structure <OUT>

Parameter	Type	Description
pcrSelect	TPML_PCR_SELECTION	information on *algID*, PCR selected and digest
pcrDigest	TPM2B_DIGEST	digest of the selected PCR using the hash of the signing key

11.12.2 TPMS_COMMAND_AUDIT_INFO

This is the *attested* data for TPM2_GetCommandAuditDigest().

Table 113 — Definition of TPMS_COMMAND_AUDIT_INFO Structure <OUT>

Parameter	Type	Description
auditCounter	UINT64	the monotonic audit counter
digestAlg	TPM_ALG_ID	hash algorithm used for the command audit
auditDigest	TPM2B_DIGEST	the current value of the audit digest
commandDigest	TPM2B_DIGEST	digest of the command codes being audited using *digestAlg*

11.12.3 TPMS_SESSION_AUDIT_INFO

This is the *attested* data for TPM2_GetSessionAuditDigest().

Table 114 — Definition of TPMS_SESSION_AUDIT_INFO Structure <OUT>

Parameter	Type	Description
exclusiveSession	TPMI_YES_NO	current exclusive status of the session TRUE if all of the commands recorded in the *sessionDigest* were executed without any intervening TPM command that did not use this transport session
sessionDigest	TPM2B_DIGEST	the current value of the session audit digest

11.12.4 TPMS_CREATION_INFO

This is the *attested* data for TPM2_CertifyCreation().

Table 115 — Definition of TPMS_CREATION_INFO Structure <OUT>

Parameter	Type	Description
objectName	TPM2B_NAME	Name of the object
creationHash	TPM2B_DIGEST	*creationHash*

11.12.5 TPMS_NV_CERTIFY_INFO

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

Table 116 — Definition of TPMS_NV_CERTIFY_INFO Structure <OUT>

Parameter	Type	Description
indexName	TPM2B_NAME	Name of the NV Index
offset	UINT16	the *offset* parameter of TPM2_NV_Certify()
nvContents	TPM2B_MAX_NV_BUFFER	contents of the NV Index

11.12.6 TPMI_ST_ATTEST

Table 117 — Definition of (TPM_ST) TPMI_ST_ATTEST Type <OUT>

Value	Description
TPM_ST_ATTEST_CERTIFY	generated by TPM2_Certify()
TPM_ST_ATTEST_QUOTE	generated by TPM2_Quote()
TPM_ST_ATTEST_SESSION_AUDIT	generated by TPM2_GetSessionAuditDigest()
TPM_ST_ATTEST_COMMAND_AUDIT	generated by TPM2_GetCommandAuditDigest()
TPM_ST_ATTEST_TIME	generated by TPM2_GetTime()
TPM_ST_ATTEST_CREATION	generated by TPM2_CertifyCreation()
TPM_ST_ATTEST_NV	generated by TPM2_NV_Certify()

11.12.7 TPMU_ATTEST

Table 118 — Definition of TPMU_ATTEST Union <OUT>

Parameter	Type	Selector
certify	TPMS_CERTIFY_INFO	TPM_ST_ATTEST_CERTIFY
creation	TPMS_CREATION_INFO	TPM_ST_ATTEST_CREATION
quote	TPMS_QUOTE_INFO	TPM_ST_ATTEST_QUOTE
commandAudit	TPMS_COMMAND_AUDIT_INFO	TPM_ST_ATTEST_COMMAND_AUDIT
sessionAudit	TPMS_SESSION_AUDIT_INFO	TPM_ST_ATTEST_SESSION_AUDIT
time	TPMS_TIME_ATTEST_INFO	TPM_ST_ATTEST_TIME
nv	TPMS_NV_CERTIFY_INFO	TPM_ST_ATTEST_NV

11.12.8 TPMS_ATTEST

This structure is used on each TPM-generated signed structure. The signature is over this structure.

When the structure is signed by a key in the Storage hierarchy, the values of *clockInfo.resetCount*, *clockInfo.restartCount*, and *firmwareVersion* are obfuscated with a per-key obfuscation value.

Table 119 — Definition of TPMS_ATTEST Structure <OUT>

Parameter	Type	Description
magic	TPM_GENERATED	the indication that this structure was created by a TPM (always TPM_GENERATED_VALUE)
type	TPMI_ST_ATTEST	type of the attestation structure
qualifiedSigner	TPM2B_NAME	Qualified Name of the signing key
extraData	TPM2B_DATA	external information supplied by caller
clockInfo	TPMS_CLOCK_INFO	*Clock*, *resetCount*, *restartCount*, and *Safe*
firmwareVersion	UINT64	TPM-vendor-specific field identifying the firmware on the TPM
[type]attested	TPMU_ATTEST	the type-specific attestation information
NOTE Regarding extraData, 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 ISO/IEC 11889.

11.12.9 TPM2B_ATTEST

This sized buffer to contain the signed structure. The *attestationData* is the signed portion of the structure. The *size* parameter is not signed.

Table 120 — Definition of TPM2B_ATTEST Structure <OUT>

Parameter	Type	Description
size	UINT16	size of the *attestationData* structure
attestationData[size]{:sizeof(TPMS_ATTEST)}	BYTE	the signed structure

11.13 Authorization Structures

11.13.1 Introduction

The structures in clause 11.13 are used for all authorizations. One or more of these structures will be present in a command or response that has a tag of TPM_ST_SESSIONS.

11.13.2 TPMS_AUTH_COMMAND

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

Table 121 — Definition of TPMS_AUTH_COMMAND Structure <IN>

Parameter	Type	Description
sessionHandle	TPMI_SH_AUTH_SESSION+	the session handle
nonce	TPM2B_NONCE	the session nonce, may be the Empty Buffer
sessionAttributes	TPMA_SESSION	the session attributes
hmac	TPM2B_AUTH	either an HMAC, a password, or an EmptyAuth

11.13.3 TPMS_AUTH_RESPONSE

This is the format for each of the authorizations in the session area of the response. If the TPM returns TPM_RC_SUCCESS, then the session area of the response contains the same number of authorizations as the command and the authorizations are in the same order.

Table 122 — Definition of TPMS_AUTH_RESPONSE Structure <OUT>

Parameter	Type	Description
nonce	TPM2B_NONCE	the session nonce, may be the Empty Buffer
sessionAttributes	TPMA_SESSION	the session attributes
Hmac	TPM2B_AUTH	either an HMAC, a password, or an EmptyAuth

12 Algorithm Parameters and Structures

12.1 Symmetric

12.1.1 Introduction

Clause 12.1 defines the parameters and structures for describing symmetric algorithms.

12.1.2 TPMI_AES_KEY_BITS

This interface type defines the supported sizes for an AES key. This type is used to allow the unmarshaling routine to generate the proper validation code for the supported key sizes. An implementation that supports different key sizes would have a different set of selections.

When used in TPM2_StartAuthSession(), the mode parameter shall be TPM_ALG_CFB.

NOTE 1 Key size is expressed in bits.

NOTE 2 The definition for AES_KEY_SIZES_BITS used in the reference implementation is found in Annex B

Table 123 — Definition of {AES} (TPM_KEY_BITS) TPMI_AES_KEY_BITS Type

Parameter	Description
$AES_KEY_SIZES_BITS	number of bits in the key
#TPM_RC_VALUE	error when key size is not supported

12.1.3 TPMI_SM4_KEY_BITS

This interface type defines the supported sizes for an SM4 key. This type is used to allow the unmarshaling routine to generate the proper validation code for the supported key sizes. An implementation that supports different key sizes would have a different set of selections.

NOTE SM4 only supports a key size of 128 bits.

Table 124 — Definition of {SM4} (TPM_KEY_BITS) TPMI_SM4_KEY_BITS Type

Parameter	Description
$SM4_KEY_SIZES_BITS	number of bits in the key
#TPM_RC_VALUE	

12.1.4 TPMI_CAMELLIA KEY_BITS

This interface type defines the supported sizes for a CAMELLIA key. This type is used to allow the unmarshaling routine to generate the proper validation code for the supported key sizes. An implementation that supports different key sizes would have a different set of selections.

Table 125 — Definition of {CAMELLIA} (TPM_KEY_BITS) TPMI_CAMELLIA_KEY_BITS Type

Parameter	Description
$CAMELLIA_KEY_SIZES_BITS	number of bits in the key
#TPM_RC_VALUE	

12.1.5 TPMU_SYM_KEY_BITS

This union is used to collect the symmetric encryption key sizes.

The *xor* entry is a hash algorithms selector and not a key size in bits. This overload is used in order to avoid an additional level of indirection with another union and another set of selectors.

The *xor* entry is only selected in a TPMT_SYM_DEF, which is used to select the parameter encryption value.

Table 126 — Definition of TPMU_SYM_KEY_BITS Union

Parameter	Type	Selector	Description
aes	TPMI_AES_KEY_BITS	TPM_ALG_AES	
SM4	TPMI_SM4_KEY_BITS	TPM_ALG_SM4	
CAMELLIA	TPMI_CAMELLIA_KEY_BITS	TPM_ALG_CAMELLIA	
sym	TPM_KEY_BITS		when selector may be any of the symmetric block ciphers
xor	TPMI_ALG_HASH	TPM_ALG_XOR	overload for using *xor*
null		TPM_ALG_NULL	

12.1.6 TPMU_SYM_MODE

This union allows the mode value in a TPMT_SYM_DEF or TPMT_SYM_DEF_OBJECT to be empty.

Table 127 — Definition of TPMU_SYM_MODE Union

Parameter	Type	Selector	Description
aes	TPMI_ALG_SYM_MODE	TPM_ALG_AES	
SM4	TPMI_ALG_SYM_MODE	TPM_ALG_SM4	
CAMELLIA	TPMI_ALG_SYM_MODE	TPM_ALG_CAMELLIA	
sym	TPMI_ALG_SYM_MODE		when selector may be any of the symmetric block ciphers
xor		TPM_ALG_XOR	no mode selector
null		TPM_ALG_NULL	no mode selector

12.1.7 TPMU_SYM_DETAILS

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

NOTE The "x" character in the table title will suppress generation of this type as the parser is not, at this time, able to generate the proper values (a union of all empty data types). When an algorithm is added that requires additional parameterization, the Type column will contain a value and the "x" may be removed.

Table 128 —xDefinition of TPMU_SYM_DETAILS Union

Parameter	Type	Selector	Description
aes		TPM_ALG_AES	
SM4		TPM_ALG_SM4	
CAMELLIA		TPM_ALG_CAMELLIA	
sym			when selector may be any of the symmetric block ciphers
xor		TPM_ALG_XOR	
null		TPM_ALG_NULL	

12.1.8 TPMT_SYM_DEF

The TPMT_SYM_DEF structure is used to select an algorithm to be used for parameter encryption in those cases when different symmetric algorithms may be selected.

Table 129 — Definition of TPMT_SYM_DEF Structure

Parameter	Type	Description
algorithm	+TPMI_ALG_SYM	indicates a symmetric algorithm
[algorithm]keyBits	TPMU_SYM_KEY_BITS	a supported key size
[algorithm]mode	TPMU_SYM_MODE	the mode for the key
//[algorithm]details	TPMU_SYM_DETAILS	contains additional algorithm details
NOTE [algorithm]details is commented out at this time as the parser might not produce the proper code for a union if none of the selectors produces any data.		

12.1.9 TPMT_SYM_DEF_OBJECT

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

Table 130 — Definition of TPMT_SYM_DEF_OBJECT Structure

Parameter	Type	Description
algorithm	+TPMI_ALG_SYM_OBJECT	selects a symmetric block cipher
[algorithm]keyBits	TPMU_SYM_KEY_BITS	the key size
[algorithm]mode	TPMU_SYM_MODE	default mode
//[algorithm]details	TPMU_SYM_DETAILS	contains the additional algorithm details, if any
NOTE	[algorithm]details is commented out at this time as the parser might not produce the proper code for a union if none of the selectors produces any data.	

12.1.10 TPM2B_SYM_KEY

This structure is used to hold a symmetric key in the sensitive area of an asymmetric object.

The number of bits in the key is in *keyBits* in the public area. When *keyBits* is not an even multiple of 8 bits, the unused bits of *buffer* will be the most significant bits of *buffer*[0] and *size* will be rounded up to the number of octets required to hold all bits of the key.

Table 131 — 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

12.1.11 TPMS_SYMCIPHER_PARMS

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

Table 132 — Definition of TPMS_SYMCIPHER_PARMS Structure

Parameter	Type	Description
sym	TPMT_SYM_DEF_OBJECT	a symmetric block cipher

12.1.12 TPM2B_SENSITIVE_DATA

This buffer holds the secret data of a data object. It can hold as much as 128 octets of data.

MAX_SYM_DATA shall be 128.

NOTE A named value rather than a numeric is used to make coding clearer. A numeric value does not indicate the reason that it has the specific value that is has.

Table 133 — Definition of TPM2B_SENSITIVE_DATA Structure

Parameter	Type	Description
size	UINT16	
buffer[size]{: MAX_SYM_DATA}	BYTE	the keyed hash private data structure

12.1.13 TPMS_SENSITIVE_CREATE

This structure defines the values to be placed in the sensitive area of a created object. This structure is only used within a TPM2B_SENSITIVE_CREATE structure.

NOTE When sent to the TPM or unsealed, data is usually encrypted using parameter encryption.

If *data.size* is not zero, and the object is not a *keyedHash*, *data.size* must match the size indicated in the *keySize* 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 134 — Definition of TPMS_SENSITIVE_CREATE Structure <IN>

Parameter	Type	Description
userAuth	TPM2B_AUTH	the USER auth secret value
data	TPM2B_SENSITIVE_DATA	data to be sealed

12.1.14 TPM2B_SENSITIVE_CREATE

This structure contains the sensitive creation data in a sized buffer. This structure is defined so that both the *userAuth* and *data* values of the TPMS_SENSITIVE_CREATE may be passed as a single parameter for parameter encryption purposes.

Table 135 — Definition of TPM2B_SENSITIVE_CREATE Structure <IN, S>

Parameter	Type	Description
size=	UINT16	size of *sensitive* in octets (may not be zero)
sensitive	TPMS_SENSITIVE_CREATE	data to be sealed or a symmetric key value.
NOTE The *userAuth* and data parameters in this buffer might both be zero length but the minimum size of the size parameter will be the sum of the size fields of the two parameters of the TPMS_SENSITIVE_CREATE.		

12.1.15 TPMS_SCHEME_SIGHASH

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

Table 136 — Definition of TPMS_SCHEME_SIGHASH Structure

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

12.1.16 TPMI_ALG_HASH_SCHEME

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

Table 137 — 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	

12.1.17 HMAC_SIG_SCHEME

Table 138 — Definition of Types for HMAC_SIG_SCHEME

Type	Name	Description
TPMS_SCHEME_SIGHASH	TPMS_SCHEME_HMAC	

12.1.18 TPMS_SCHEME_XOR

This structure is for the XOR encryption scheme.

Table 139 — 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

12.1.19 TPMU_SCHEME_HMAC

Table 140 — Definition of TPMU_SCHEME_KEYEDHASH Union <IN/OUT, S>

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	

12.1.20 TPMT_KEYEDHASH_SCHEME

This structure is used for a hash signing object.

Table 141 — 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

12.2 Asymmetric

12.2.1 Signing Schemes

12.2.1.1 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.

12.2.1.2 RSA_SIG_SCHEMES

These are the RSA schemes that only need a hash algorithm as a scheme parameter.

For the TPM_ALG_RSAPSS signing scheme, the same hash algorithm is used for digesting TPM-generated data (an attestation structure) and in the KDF used for the masking operation. The salt size is always the largest salt value that will fit into the available space.

Table 142 — Definition of {RSA} Types for RSA_SIG_SCHEMES

Type	Name	Description
TPMS_SCHEME_SIGHASH	TPMS_SCHEME_RSASSA	
TPMS_SCHEME_SIGHASH	TPMS_SCHEME_RSAPSS	

12.2.1.3 ECC_SIG_SCHEMES

These are the ECC schemes that only need a hash algorithm as a controlling parameter.

Table 143 — Definition of {ECC} Types for ECC_SIG_SCHEMES

Type	Name	Description
TPMS_SCHEME_SIGHASH	TPMS_SCHEME_ECDSA	
TPMS_SCHEME_SIGHASH	TPMS_SCHEME_SM2	
TPMS_SCHEME_SIGHASH	TPMS_SCHEME_ECSCHNORR	

12.2.1.4 TPMS_SCHEME_ECDAA

Table 144 — Definition of {ECC} TPMS_SCHEME_ECDAA Structure

Parameter	Type	Description
hashAlg	TPMI_ALG_HASH	the hash algorithm used to digest the message
count	UINT16	the counter value that is used between TPM2_Commit() and the sign operation

12.2.1.5 TPMU_SIG_SCHEME

Table 145 — Definition of TPMU_SIG_SCHEME Union <IN/OUT, S>

Parameter	Type	Selector	Description
rsassa	TPMS_SCHEME_RSASSA	TPM_ALG_RSASSA	the RSASSA-PKCS1v1_5 scheme
rsapss	TPMS_SCHEME_RSAPSS	TPM_ALG_RSAPSS	the RSASSA-PSS scheme
ecdsa	TPMS_SCHEME_ECDSA	TPM_ALG_ECDSA	the ECDSA scheme
sm2	TPMS_SCHEME_SM2	TPM_ALG_SM2	ECDSA from SM2
ecdaa	TPMS_SCHEME_ECDAA	TPM_ALG_ECDAA	the ECDAA scheme
ecSchnorr	TPMS_SCHEME_ECSCHNORR	TPM_ALG_ECSCHNORR	the EC Schnorr
hmac	TPMS_SCHEME_HMAC	TPM_ALG_HMAC	the HMAC scheme
any	TPMS_SCHEME_SIGHASH		selector that allows access to digest for any signing scheme
null		TPM_ALG_NULL	no scheme or default

12.2.1.6 TPMT_SIG_SCHEME

Table 146 — Definition of TPMT_SIG_SCHEME Structure

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

12.2.2 Encryption Schemes

12.2.2.1 Introduction

These structures are used to indicate the hash algorithm used for the encrypting process. These schemes would appear in an object's public area.

12.2.2.2 TPMS_SCHEME_OAEP

Table 147 — Definition of {RSA} TPMS_SCHEME_OAEP Structure

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

12.2.2.3 TPMS_SCHEME_ECDH

For ECDH, **KDFe** is used for the key derivation function that so only a hash algorithm is needed to complete the definition.

Table 148 — Definition of {ECC} TPMS_SCHEME_ECDH Structure

Parameter	Type	Description
hashAlg	TPMI_ALG_HASH	the hash algorithm used in the KDF

12.2.3 Key Derivation Schemes

12.2.3.1 Introduction

These structures are used to define the key derivation for symmetric secret sharing using asymmetric methods. A secret sharing scheme is required in any asymmetric key with the *decrypt* attribute SET.

These schemes would appear in an object's public area and in commands where the secret sharing scheme is variable.

Each scheme includes a symmetric algorithm and a KDF selection.

12.2.3.2 TPMS_SCHEME_MGF1

Table 149 — Definition of TPMS_SCHEME_MGF1 Structure

Parameter	Type	Description
hashAlg	TPMI_ALG_HASH	the hash algorithm used in the KDF

12.2.3.3 TPMS_SCHEME_KDF1_SP800_56a

Table 150 — Definition of {ECC} TPMS_SCHEME_KDF1_SP800_56a Structure

Parameter	Type	Description
hashAlg	TPMI_ALG_HASH	the hash algorithm used in the KDF

12.2.3.4 TPMS_SCHEME_KDF2

Table 151 — Definition of TPMS_SCHEME_KDF2 Structure

Parameter	Type	Description
hashAlg	TPMI_ALG_HASH	the hash algorithm used in the KDF

12.2.3.5 TPMS_SCHEME_KDF1_SP800_108

Table 152 — Definition of TPMS_SCHEME_KDF1_SP800_108 Structure

Parameter	Type	Description
hashAlg	TPMI_ALG_HASH	the hash algorithm used in the KDF

12.2.3.6 TPMU_KDF_SCHEME

Table 153 — Definition of TPMU_KDF_SCHEME Union <IN/OUT, S>

Parameter	Type	Selector	Description
mgf1	TPMS_SCHEME_MGF1	TPM_ALG_MGF1	
kdf1_SP800_56a	TPMS_SCHEME_KDF1_SP800_56a	TPM_ALG_KDF1_SP800_56a	
kdf2	TPMS_SCHEME_KDF2	TPM_ALG_KDF2	
kdf1_sp800_108	TPMS_SCHEME_KDF1_SP800_108	TPM_ALG_KDF1_SP800_108	
null		TPM_ALG_NULL	

12.2.3.7 TPMT_KDF_SCHEME

Table 154 — Definition of TPMT_KDF_SCHEME Structure

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

12.2.3.8 TPMI_ALG_ASYM_SCHEME

List of all of the scheme types for any asymmetric algorithm. This is used to define the TPMT_ASYM_SCHEME.

Table 155 — Definition of (TPM_ALG_ID) TPMI_ALG_ASYM_SCHEME Type <>

Values	Comments
TPM_ALG_RSASSA	list of the allowed values
TPM_ALG_RSAPSS	
TPM_ALG_RSAES	
TPM_ALG_OAEP	
TPM_ALG_ECDSA	
TPM_ALG_SM2	
TPM_ALG_ECDAA	
TPM_ALG_ECDH	
+TPM_ALG_NULL	
#TPM_RC_VALUE	

12.2.3.9 TPMU_ASYM_SCHEME

This union of all asymmetric schemes is used in each of the asymmetric scheme structures. The actual scheme structure is defined by the interface type used for the selector.

EXAMPLE The TPMT_RSA_SCHEME structure uses the TPMU_ASYM_SCHEME union but the selector type is TPMI_ALG_RSA_SCHEME. This means that the only elements of the union that can be selected for the TPMT_RSA_SCHEME are those that are in TPMI_RSA_SCHEME.

Table 156 — Definition of TPMU_ASYM_SCHEME Union

Parameter	Type	Selector	Description
rsassa	TPMS_SCHEME_RSASSA	TPM_ALG_RSASSA	the RSASSA-PKCS1-v1_5 scheme
rsapss	TPMS_SCHEME_RSAPSS	TPM_ALG_RSAPSS	the RSASSA-PSS scheme
rsaes		TPM_ALG_RSAES	the RSAES-PKCS1-v1_5 scheme
oaep	TPMS_SCHEME_OAEP	TPM_ALG_OAEP	the RSAES_OAEP scheme
ecdsa	TPMS_SCHEME_ECDSA	TPM_ALG_ECDSA	an ECDSA scheme
sm2	TPMS_SCHEME_SM2	TPM_ALG_SM2	sign or key exchange from SM2
ecdaa	TPMS_SCHEME_ECDAA	TPM_ALG_ECDAA	an ECDAA scheme
ecSchnorr	TPMS_SCHEME_ECSCHNORR	TPM_ALG_ECSCHNORR	elliptic curve Schnorr signature
ecdh	TPMS_SCHEME_ECDH	TPM_ALG_ECDH	
anySig	TPMS_SCHEME_SIGHASH		
null		TPM_ALG_NULL	no scheme or default This selects the NULL Signature.

12.2.3.10 TPMT_ASYM_SCHEME

This structure is defined to allow overlay of all of the schemes for any asymmetric object. This structure is not sent on the interface.

Table 157 — Definition of TPMT_ASYM_SCHEME Structure <>

Parameter	Type	Description
scheme	+TPMI_ALG_ASYM_SCHEME	scheme selector
[scheme]details	TPMU_ASYM_SCHEME	scheme parameters

12.2.4 RSA

12.2.4.1 TPMI_ALG_RSA_SCHEME

The list of values that may appear in the scheme parameter of a TPMS_RSA_PARMS structure.

Table 158 — Definition of (TPM_ALG_ID) {RSA} TPMI_ALG_RSA_SCHEME Type

Values	Comments
TPM_ALG_RSASSA	list of the allowed values
TPM_ALG_RSAPSS	
TPM_ALG_RSAES	
TPM_ALG_OAEP	
+TPM_ALG_NULL	
#TPM_RC_VALUE	

12.2.4.2 TPMT_RSA_SCHEME

Table 159 — Definition of {RSA} TPMT_RSA_SCHEME Structure

Parameter	Type	Description
scheme	+TPMI_ALG_RSA_SCHEME	scheme selector
[scheme]details	TPMU_ASYM_SCHEME	scheme parameters

12.2.4.3 TPMI_ALG_RSA_DECRYPT

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

Table 160 — 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	

12.2.4.4 TPMT_RSA_DECRYPT

Table 161 — Definition of {RSA} TPMT_RSA_DECRYPT Structure

Parameter	Type	Description
scheme	+TPMI_ALG_RSA_DECRYPT	scheme selector
[scheme]details	TPMU_ASYM_SCHEME	scheme parameters

12.2.4.5 TPM2B_PUBLIC_KEY_RSA

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

NOTE The reference implementation only supports key sizes of 1,024 and 2,048 bits.

Table 162 — 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

12.2.4.6 TPMI_RSA_KEY_BITS

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

NOTE 1 An implementation is can provide limited support for smaller RSA key sizes. That is, a TPM might 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 let the TPM to validate signatures using the smaller key but would prevent the TPM from using the smaller key size for any other purpose.

NOTE 2 The definition for RSA_KEY_SIZES_BITS used in the reference implementation is found in Annex B

Table 163 — 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

12.2.4.7 TPM2B_PRIVATE_KEY_RSA

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

NOTE All primes need to have exactly half the number of significant bits as the public modulus, and the square of each prime needs to have the same number of significant bits as the public modulus.

Table 164 — Definition of {RSA} TPM2B_PRIVATE_KEY_RSA Structure

Parameter	Type	Description
size	UINT16	
buffer[size]{:MAX_RSA_KEY_BYTES/2}	BYTE	

12.2.5 ECC

12.2.5.1 TPM2B_ECC_PARAMETER

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

Table 165 — Definition of {ECC} TPM2B_ECC_PARAMETER Structure

Parameter	Type	Description
size	UINT16	size of *buffer*
buffer[size] {:MAX_ECC_KEY_BYTES}	BYTE	the parameter data

12.2.5.2 TPMS_ECC_POINT

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

Table 166 — Definition of {ECC} TPMS_ECC_POINT Structure

Parameter	Type	Description
x	TPM2B_ECC_PARAMETER	X coordinate
y	TPM2B_ECC_PARAMETER	Y coordinate

12.2.5.3 TPM2B_ECC_POINT

This structure is defined to allow a point to be a single sized parameter so that it may be encrypted.

NOTE If the point is to be omitted, the X and Y coordinates need to be individually set to Empty Buffers. The minimum value for size will be four. It is checked indirectly by unmarshaling of the TPMS_ECC_POINT. If the type of *point* were BYTE, then *size* could have been zero. However, this would complicate the process of marshaling the structure.

Table 167 — Definition of {ECC} TPM2B_ECC_POINT Structure

Parameter	Type	Description
size=	UINT16	size of the remainder of this structure
point	TPMS_ECC_POINT	coordinates
#TPM_RC_SIZE		error returned if the unmarshaled size of *point* is not exactly equal to *size*

12.2.5.4 TPMI_ALG_ECC_SCHEME

Table 168 — Definition of (TPM_ALG_ID) {ECC} TPMI_ALG_ECC_SCHEME Type

Values	Comments
TPM_ALG_ECDSA	these are the selections allowed for an ECC key
TPM_ALG_SM2	
TPM_ALG_ECDAA	
TPM_ALG_ECSCHNORR	
TPM_ALG_ECDH	
+TPM_ALG_NULL	
#TPM_RC_SCHEME	

12.2.5.5 TPMI_ECC_CURVE

The ECC curves implemented by the TPM.

NOTE The definition of ECC_CURVES used in the reference implementation is found in Annex B

Table 169 — Definition of {ECC} (TPM_ECC_CURVE) TPMI_ECC_CURVE Type

Parameter	Description
$ECC_CURVES	the list of implemented curves
#TPM_RC_CURVE	error when curve is not supported

12.2.5.6 TPMT_ECC_SCHEME

Table 170 — Definition of (TPMT_SIG_SCHEME) {ECC} TPMT_ECC_SCHEME Structure

Parameter	Type	Description
scheme	+TPMI_ALG_ECC_SCHEME	scheme selector
[scheme]details	TPMU_ASYM_SCHEME	scheme parameters

12.2.5.7 TPMS_ALGORITHM_DETAIL_ECC

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

Table 171 — Definition of {ECC} TPMS_ALGORITHM_DETAIL_ECC Structure <OUT>

Parameter	Type	Description
curveID	TPM_ECC_CURVE	identifier for the curve
keySize	UINT16	Size in bits of the key
kdf	TPMT_KDF_SCHEME	the default KDF and hash algorithm used in secret sharing operations
sign	TPMT_ECC_SCHEME+	If not TPM_ALG_NULL, this is the mandatory signature scheme that is required to be used with this curve.
p	TPM2B_ECC_PARAMETER	Fp (the modulus)
a	TPM2B_ECC_PARAMETER	coefficient of the linear term in the curve equation
b	TPM2B_ECC_PARAMETER	constant term for curve equation
gX	TPM2B_ECC_PARAMETER	x coordinate of base point G
gY	TPM2B_ECC_PARAMETER	y coordinate of base point G
n	TPM2B_ECC_PARAMETER	order of G
h	TPM2B_ECC_PARAMETER	cofactor (a size of zero indicates a cofactor of 1)

12.3 Signatures

12.3.1 TPMS_SIGNATURE_RSASSA

Table 172 — Definition of {RSA} TPMS_SIGNATURE_RSASSA 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.

12.3.2 TPMS_SIGNATURE_RSAPSS

When the TPM generates a PSS signature, the salt size is the largest size allowed by the key and hash combination.

EXAMPLE For a 2,048-bit public modulus key and SHA1 hash, the salt size is 256 – 20 – 2 = 234 octets.

NOTE While this is significantly larger than needed from a security perspective, it avoids issues of whether a particular size of salt value is sufficient.

