

BS ISO/IEC 11889-2:2015



BSI Standards Publication

Information technology — Trusted Platform Module Library

Part 2: Structures

bsi.

...making excellence a habit.
AVA Documents™
Document Sharing Hub

National foreword

This British Standard is the UK implementation of ISO/IEC 11889-2:2015. It supersedes BS ISO/IEC 11889-2:2009, which will be withdrawn on 1 January 2026.

BS ISO/IEC 11889-2:2009 remains current until 1 January 2026 due to its extensive implementation by existing products and the request from a number of National Bodies that it continues to remain available in order to support existing deployments that currently incorporate its provisions.

The UK participation in its preparation was entrusted to Technical Committee IST/33, IT - Security techniques.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© The British Standards Institution 2016.
Published by BSI Standards Limited 2016

ISBN 978 0 580 86950 1

ICS 35.040

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 January 2016.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------

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)

AVA Documents
Document Sharing Hub
© ISO/IEC 2015



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2015, Published in Switzerland

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

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

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	<i>Clock</i>	101
11.11.3	<i>ResetCount</i>	101
11.11.4	<i>RestartCount</i>	102
11.11.5	<i>Safe</i>	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 <i>sequence</i>	143
15.6.2 <i>savedHandle</i>	144
15.6.3 <i>hierarchy</i>	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 <i>proof</i> 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_ECDSA 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 <i>number</i>
[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 <i>number</i>
[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 <i>value1</i> , which is a value limited to be between 20 and 25 inclusive, array1 is an array that may have between 20 and 25 octets. This is not the preferred way to indicate the upper limit for an array as it does not indicate the upper bound of the size.
array2[value4][:25]	BYTE	an array that may have between 20 and 25 octets This arrangement is used to allow the automatic code generation to allocate 25 octets to store the largest array2 that can be unmarshaled. The code generation can determine from this expression that <i>value4</i> shall have a value of 25 or less. From the definition of <i>value4</i> above, it can determine that <i>value4</i> must have a value of at least 20.
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 <i>size</i> . That is, the number of bytes in the input buffer used to successfully unmarshal <i>someStructure</i> must be the same as <i>size</i> .
someStructure	A_STRUCT	a structure to be unmarshaled The size of the structure is computed when it is unmarshaled. Because an “=” is present on the definition of <i>size</i> , the TPM is required to validate that the unmarshaled size exactly matches <i>size</i> .
someData	UINT32	a value

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 <i>commandCode</i> . 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 <i>responseCode</i> . 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 <i>authorizationSize</i> parameter in a command
UINT32	TPM_PARAMETER_SIZE	the <i>parameterSizeSet</i> 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 00₁₆ – 7F FF₁₆. Other structure tags will be in the range 80 00₁₆ – FF FF₁₆.

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 C1₁₆ – 00 C6₁₆ 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	(Classification) The allowed values are: A – Assigned S – TCG Standard L – TCG Legacy
Dep	(Dependent) 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_ECDA	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.

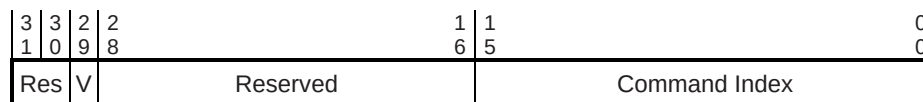


Figure 1 — Command Format

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 <i>commandCode</i>
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 <i>Clock</i> .	
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 00₁₆ and the low-order 12 bits (11:00) will contain the response code.

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

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

- a TPM_ST (UINT16) with a value of TPM_TAG_RSP_COMMAND or TPM_ST_NO_SESSIONS, followed by
- a UINT32 (*responseSize*) with a value of 10, followed by
- a UINT32 containing a response code with a value other than TPM_RC_SUCCESS.

Commands defined in 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 1E₁₆). 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	1	1	0	0	0	0	0	0	0	0	0	0
	1	0	9	8	7	6	5	4	3	2	1	0
	S	T	r	V	F	E						

Figure 2 — Format-Zero Response Codes