Trusted Platform Module Library

Part 2: Structures

Family "2.0"

Level 00 Revision 01.38

September 29, 2016

Contact: admin@trustedcomputinggroup.org

TCG

TCG Published

Licenses and Notices

CONTENTS

Tables

Figures

Trusted Platform Module Library

Part 2: Structures

1 Scope

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

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

2 Terms and definitions

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

3 Symbols and abbreviated terms

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

4 Notation

4.1 Introduction

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

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

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

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

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

4.2 Named Constants

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

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

The table in 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
```


4.3 Data Type Aliases (typedefs)

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

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

EXAMPLE 1

Table xx — Definition of Types for Some Purpose

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

EXAMPLE 2

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

4.4 Enumerations

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

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

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

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

EXAMPLE 1

Table xx — Definition of (UINT16) CARD_SUIT Values

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

EXAMPLE 2

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


4.5 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 this specification.

EXAMPLE 2

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

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

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

EXAMPLE 3

Table xx — Defines for MY Algorithm Constants

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

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

EXAMPLE 4

Table xx — Definition of (UINT16) MY_KEY_BITS Type

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

EXAMPLE 5

Table xx — Definition of (UINT16) MY_KEY_BITS Type

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

4.6 Arrays

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

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

EXAMPLE 1

Table xx — Definition of A_STRUCT Structure

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

EXAMPLE 2

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

4.7 Structure Definitions

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

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

EXAMPLE 1

Table xx — Definition of SIMPLE_STRUCTURE Structure

Parameter	Type	Description
tag	TPM_ST	
value1	INT32	
value2	INT32	

EXAMPLE 2

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

EXAMPLE 3

```
TPM_RC SIMPLE_STRUCTURE_Unmarshal(SIMPLE_STRUCTURE *target, BYTE **buffer, INT32 *size)
{
    TPM_RC      rc;
    // If unmarshal of tag succeeds
    rc = TPM_ST_Unmarshal((TPM_ST *)&(target->tag), buffer, size
    If(rc  == TPM_RC_SUCCESS)
    {
         // then unmarshal value1,
        rc = INT32_Unmarshal((INT32 *)&(target->value1, buffer, size);
        // and if that succeeds...
        if(rc  == TPM_RC_SUCCESS)
        {
            // then unmarshal the value 2
            rc = INT32_Unmarshal((INT32 *)&(target->value2, buffer, size);
        }
    }
    return rc;
}
```


A table may have a term in {}. This indicates that the table is conditionally compiled. It is commonly used when a table's inclusion is based on the implementation of a cryptographic algorithm. See, for example, Table 162 — Definition of (TPM_ALG_ID) {RSA} TPMI_ALG_RSA_DECRYPT Type, which is dependent on the RSA algorithm.

4.8 Conditional Types

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

EXAMPLE 1 Table 60 defining TPMI_ALG_HASH indicates that TPM_ALG_NULL is a conditional type. TPMI_ALG_HASH is a member of the TPMS_SCHEME_XOR structure and that reference is TPMI_ALG_HASH+, indicating that TPM_ALG_NULL is an allowed value for hashAlg. TPMI_ALG_HASH is also referenced in TPMS_PCR_SELECTION. In that structure the TPMI_ALG_HASH does not have an appended “+”, so TPM_ALG_NULL would not be an allowed value for hash.

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

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

EXAMPLE 2 Table 131 — Definition of TPMT_SYM_DEF_OBJECT Structure defines the TPMT_SYM_DEF_OBJECT. The algorithm element of that structure is a TPMI_ALG_SYM_OBJECT with a “+” prepended. This means that when a TPMT_SYM_DEF_OBJECT is referenced, the reference may have an appended “+” as it does in the definition of the symmetric parameter of TPMS_ASYM_PARMS. The “+” in TPMA_ASYM_PARMS means that the algorithm parameter in the TPMT_SYM_DEF_OBJECT may have the conditional value (TPM_ALG_NULL).

EXAMPLE 3

Table xx — Definition of (CARD_SUIT) TPMI_CARD_SUIT Type

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

EXAMPLE 4

Table xx — Definition of POKER_CARD Structure

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

EXAMPLE 5

Table xx — Definition of BRIDGE_CARD Structure

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

4.9 Unions

4.9.1 Introduction

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

- union definition;
- union instance; and
- union selector definition.

4.9.2 Union Definition

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

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

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

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

EXAMPLE 1

Table xx — Definition of NUMBER_UNION Union

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

EXAMPLE 2

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

EXAMPLE 3

```
// Possible auto-generated code to unmarshal a union in Example 2 based on the
// input value of selector
TPM_RC NUMBER_UNION_Unmarshal(NUMBER_UNION *target, BYTE **buffer,
                              INT32 *size, UINT32 selector)
{
    switch (selector) {
    case BYTE_SELECT:
        return BYTE_Unmarshal((BYTE *)&(target->a_byte), buffer, size);
    case INT_SELECT:
        return INT_Unmarshal((int *)&(target->an_int), buffer, size);
    case FLOAT_SELECT:
        return FLOAT_Unmarshal((float *)&(target->a_float), buffer, size);
    case NULL_SELECT:
        return TPM_RC_SUCCESS;
}
```

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

EXAMPLE 4 Table 178 has an 'any' parameter with no selector. Any of the other union members may be cast to TPMS_SCHEME_HASH, since all begin with TPMI_ALG_HASH.

4.9.3 Union Instance

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

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

EXAMPLE 1

Table xx — Definition of STRUCTURE_WITH_UNION Structure

Parameter	Type	Description
select	NUMBER_SELECT	a value indicating the type in *number*
[select] number	NUMBER_UNION	a union as shown in 4.9.2

EXAMPLE 2

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

EXAMPLE 3

```
// Possible unmarshaling code for the structure above
TPM_RC STRUCT_WITH_UNION_Unmarshal(STRUCT_WITH_UNION *target, BYTE **buffer, INT32 *size)
{
    TPM_RC          rc;
    // Unmarshal the selector value
    rc = NUMBER_SELECT_Unmarshal((NUMBER_SELECT *)&target->select, buffer, size)
    if(rc != TPM_RC_SUCCESS)
        return rc;
    // Use the unmarshaled selector value to indicate to the union unmarshal
    // function which unmarshaling branch to follow.
    return(NUMBER_UNION_Unmarshal((NUMBER_UNION *)&(target->number),
                                  buffer, size, (UINT32)target->select);
}
```

4.9.4 Union Selector Definition

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

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

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

EXAMPLE 1

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

Name	Value	Comments
BYTE_SELECT	3	
INT_SELECT	2	
FLOAT_SELECT	1	
NULL_SELECT	0	

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

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

EXAMPLE 2

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

Values	Comments
{BYTE_SELECT, INT_SELECT}	list of allowed values

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

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

EXAMPLE 3

Table xx — Definition of JUST_INTEGERS Structure

Parameter	Type	Description
select	AN_INTEGER	a value indicating the type in *number*
[select] number	NUMBER_UNION	a union as shown in 4.9.2

4.10 Bit Field Definitions

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

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

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

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

EXAMPLE1

Table xx — Definition of (UINT32) SOME_ATTRIBUTE Bits

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

EXAMPLE 2

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

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

4.11 Parameter Limits

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

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

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

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

EXAMPLE

Table xx — Definition of B_STRUCT Structure

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

Parameter	Type	Description
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.

4.12 Algorithm Macros

4.12.1 Introduction

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

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

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

4.12.2 Algorithm Token Semantics

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

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

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

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

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

NOTE The root identifier of an algorithm is the name in the TPM_ALG_ID table with “TPM_ALG_” removed. For example TPM_ALG_SHA1 has “SHA1” as its root.

The typical use of these tokens follows.

4.12.3 Algorithm Tokens in Unions

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

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

Table A — Definition of TPMU_HA Union

Parameter	Type	Selector	Description
!ALG.H [!ALG_DIGEST_SIZE]	BYTE	TPM_ALG_!ALG	all hashes
null		TPM_ALG_NULL	

If the TCG registry only contained SHA1, SHA256, and the SM3_256 hash algorithm identifiers, then the table above would be semantically equivalent to:

Table xx — Definition of TPMU_HA Union

Parameter	Type	Selector	Description
sha1 [SHA1_DIGEST_SIZE]	BYTE	TPM_ALG_SHA1	
sha256 [SHA256_DIGEST_SIZE]	BYTE	TPM_ALG_SHA256	
sm3_256 [SM3_256_DIGEST_SIZE]	BYTE	TPM_ALG_SM3_256	
null		TPM_ALG_NULL	

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

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

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

4.12.4 Algorithm Tokens in Interface Types

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

EXAMPLE If SHA1, SHA256, and SM3_256 are the only defined hash algorithms, then an interface type to select a hash would be:

Table xx — Definition of (TPM_ALG_ID) TPMI_ALG_HASH Type

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

An equivalent table may be represented using an algorithm macro.

Table xx — Definition of (TPM_ALG_ID) TPMI_ALG_HASH Type

Values	Comments
TPM_ALG_!ALG.H	all hash algorithms defined by the TCG
+TPM_ALG_NULL	
#TPM_RC_HASH	

4.12.5 Algorithm Tokens for Table Replication

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

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

Table xx — Definition of {!ALG.S} (TPM_KEY_BITS) TPMI_!ALG_KEY_BITS Type

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

has the same meaning as:

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

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

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

Parameter	Description
$SM4_KEY_SIZES_BITS	number of bits in the key
#TPM_RC_VALUE	error when key size is not supported

4.13 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 4.11, *value4* indicates how many octets to unmarshal for *array2*. This semantic applies when the size field determines the number of octets to unmarshal. However, in some cases, the subsequent structure is self-defining. If the size precedes a parameter that is not an octet array, then the unmarshaled size of that parameter is determined by its data type. The table in Example 1 shows a structure where the size parameter would nominally indicate the number of octets in the remainder of the structure.

EXAMPLE 1

Table xx — Definition of C_STRUCT Structure

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

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

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

EXAMPLE 2

Table xx — Definition of D_STRUCT Structure

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

4.14 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>

4.15 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>

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

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

4.17 Data Alignment

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

4.18 Parameter Unmarshaling Errors

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

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

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

When an error is encountered while unmarshaling a command parameter, an error response code is returned and no command processing occurs. A table defining a data type may have response codes embedded in the table to indicate the error returned when the input value does not match the parameters of the table.

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

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

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

Table 2 — Unmarshaling Errors

Response code	Usage
TPM_RC_INSUFFICIENT	the input buffer did not contain enough octets to allow unmarshaling of the expected data type;
TPM_RC_RESERVED_BITS	a non-zero value was found in a reserved field of an attribute structure (TPMA_)
TPM_RC_SIZE	the value of a size parameter is larger or smaller than allowed
TPM_RC_VALUE	A parameter does not have one of its allowed values
TPM_RC_TAG	A parameter that should be a structure tag has a value that is not supported by the TPM

In some commands, a parameter may not be used because of various options of that command. However, the unmarshaling code is required to validate that all parameters have values that are allowed by the TPM 2.0 Part 2 definition of the parameter type even if that parameter is not used in the command actions.

5 Base Types

5.1 Primitive Types

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

NOTE The types are compatible with the C99 standard and 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

5.2 Specification Logic Value Constants

Table 4 — Defines for Logic Values

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

5.3 Miscellaneous Types

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

Table 5 — Definition of Types for Documentation Clarity

Type	Name	Description
UINT32	TPM_ALGORITHM_ID	this is the 1.2 compatible form of the TPM_ALG_ID
UINT32	TPM_MODIFIER_INDICATOR	
UINT32	TPM_AUTHORIZATION_SIZE	the *authorizationSize* parameter in a command
UINT32	TPM_PARAMETER_SIZE	the *parameterSize* parameter in a command
UINT16	TPM_KEY_SIZE	a key size in octets
UINT16	TPM_KEY_BITS	a key size in bits

6 Constants

6.1 TPM_SPEC (Specification Version Values)

These values are readable with TPM2_GetCapability().

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

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

Table 6 — Definition of (UINT32) TPM_SPEC Constants <>

Name	Value	Comments
TPM_SPEC_FAMILY	0x322E3000	ASCII “2.0” with null terminator
TPM_SPEC_LEVEL	00	the level number for the specification
TPM_SPEC_VERSION	138	the version number of the spec (001.38 * 100)
TPM_SPEC_YEAR	2016	the year of the version
TPM_SPEC_DAY_OF_YEAR	273	the day of the year (September 29, 2016)

6.2 TPM_GENERATED

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

Table 7 — Definition of (UINT32) TPM_GENERATED Constants <O>

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

6.3 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.

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

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

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

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

An algorithm shall not be assigned a value in the range $00\ C1_{16}$ – $00\ C6_{16}$ in order to prevent any overlap with the command structure tags used in TPM 1.2.

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

EXAMPLE Implementation of TPM_ALG_RSASSA requires that the RSA algorithm be implemented.

TPM_ALG_KEYEDHASH and TPM_ALG_NULL are required of all TPM implementations.

Table 8 — Legend for TPM_ALG_ID Table

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

Table 9 — Definition of (UINT16) TPM_ALG_ID Constants <IN/OUT, S>

Algorithm Name	Value	Type	Dep	C	Reference	Comments
TPM_ALG_ERROR	0x0000					should not occur
TPM_ALG_RSA	0x0001	A O		A	IETF RFC 3447	the RSA algorithm
TPM_ALG_SHA	0x0004	H		A	ISO/IEC 10118-3	the SHA1 algorithm
TPM_ALG_SHA1	0x0004	H		A	ISO/IEC 10118-3	redefinition for documentation consistency
TPM_ALG_HMAC	0x0005	H X		A	ISO/IEC 9797-2	Hash Message Authentication Code (HMAC) algorithm
TPM_ALG_AES	0x0006	S		A	ISO/IEC 18033-3	the AES algorithm with various key sizes
TPM_ALG_MGF1	0x0007	H M		A	IEEE Std 1363™-2000 IEEE Std 1363a™-2004	hash-based mask-generation function
TPM_ALG_KEYEDHASH	0x0008	H E X O S		S	TCG TPM 2.0 library specification	an encryption or signing algorithm using a keyed hash May also refer to a data object that is neither signing nor encrypting
TPM_ALG_XOR	0x000A	H S		A	TCG TPM 2.0 library specification	the XOR encryption algorithm
TPM_ALG_SHA256	0x000B	H		A	ISO/IEC 10118-3	the SHA 256 algorithm
TPM_ALG_SHA384	0x000C	H		A	ISO/IEC 10118-3	the SHA 384 algorithm
TPM_ALG_SHA512	0x000D	H		A	ISO/IEC 10118-3	the SHA 512 algorithm
TPM_ALG_NULL	0x0010			S	TCG TPM 2.0 library specification	Null algorithm
TPM_ALG_SM3_256	0x0012	H		A	GM/T 0004-2012	SM3 hash algorithm
TPM_ALG_SM4	0x0013	S		A	GM/T 0002-2012	SM4 symmetric block cipher
TPM_ALG_RSASSA	0x0014	A X	RSA	A	IETF RFC 3447	a signature algorithm defined in section 8.2 (RSASSA-PKCS1-v1_5)
TPM_ALG_RSAES	0x0015	A E	RSA	A	IETF RFC 3447	a padding algorithm defined in section 7.2 (RSAES-PKCS1-v1_5)
TPM_ALG_RSAPSS	0x0016	A X	RSA	A	IETF RFC 3447	a signature algorithm defined in section 8.1 (RSASSA-PSS)
TPM_ALG_OAEP	0x0017	A E H	RSA	A	IETF RFC 3447	a padding algorithm defined in section 7.1 (RSAES_OAEP)
TPM_ALG_ECDSA	0x0018	A X	ECC	A	ISO/IEC 14888-3	signature algorithm using elliptic curve cryptography (ECC)

Algorithm Name	Value	Type	Dep	C	Reference	Comments
TPM_ALG_ECDH	0x0019	A M	ECC	A	NIST SP800-56A	secret sharing using ECC Based on context, this can be either One-Pass Diffie-Hellman, C(1, 1, ECC CDH) defined in 6.2.2.2 or Full Unified Model C(2, 2, ECC CDH) defined in 6.1.1.2
TPM_ALG_ECDAA	0x001A	A X N	ECC	A	TCG TPM 2.0 library specification	elliptic-curve based, anonymous signing scheme
TPM_ALG_SM2	0x001B	A X	ECC	A	GM/T 0003.1–2012 GM/T 0003.2–2012 GM/T 0003.3–2012 GM/T 0003.5–2012	SM2 – depending on context, either an elliptic-curve based, signature algorithm or a key exchange protocol NOTE Type listed as signing but, other uses are allowed according to context.
TPM_ALG_ECSCHNORR	0x001C	A X	ECC	A	TCG TPM 2.0 library specification	elliptic-curve based Schnorr signature
TPM_ALG_ECMQV	0x001D	A M	ECC	A	NIST SP800-56A	two-phase elliptic-curve key exchange – C(2, 2, ECC MQV) section 6.1.1.4
TPM_ALG_KDF1_SP800_56A	0x0020	H M	ECC	A	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		A	NIST SP800-108	a key derivation method Section 5.1 KDF in Counter Mode
TPM_ALG_ECC	0x0023	A O		A	ISO/IEC 15946-1	prime field ECC
TPM_ALG_SYMCIPHER	0x0025	O S		A	TCG TPM 2.0 library specification	the object type for a symmetric block cipher
TPM_ALG_CAMELLIA	0x0026	S		A	ISO/IEC 18033-3	Camellia is symmetric block cipher. The Camellia algorithm with various key sizes
TPM_ALG_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.

Algorithm Name	Value	Type	Dep	C	Reference	Comments
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.
TPM_ALG_ECB	0x0044	S E		A	ISO/IEC 10116	Electronic Codebook mode – if implemented, all symmetric block ciphers (S type) implemented shall be capable of using this mode. NOTE This mode is not recommended for uses unless the key is frequently rotated such as in video codecs
reserved	0x00C1 through 0x00C6					0x00C1 – 0x00C6 are reserved to prevent any overlap with the command structure tags used in TPM 1.2
reserved	0x8000 through 0xFFFF					reserved for other structure tags

6.4 TPM_ECC_CURVE

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

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

Table 10 — Definition of (UINT16) {ECC} TPM_ECC_CURVE Constants <IN/OUT, S>

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

6.5 TPM_CC (Command Codes)

6.5.1 Format

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

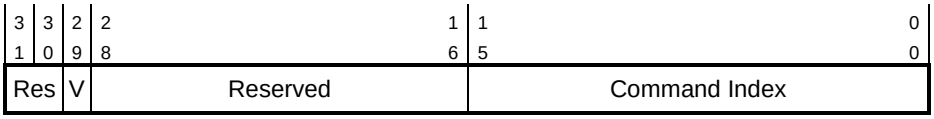

Figure 1 — Command Format

Table 11 — TPM Command Format Fields Description

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

6.5.2 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 TPM 2.0 Part 3 are definitive.

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

Name	Command Code	Comments
TPM_CC_FIRST	0x0000011F	Compile variable. May decrease based on implementation.
TPM_CC_NV_UndefineSpaceSpecial	0x0000011F	
TPM_CC_EvictControl	0x00000120	
TPM_CC_HierarchyControl	0x00000121	
TPM_CC_NV_UndefineSpace	0x00000122	
TPM_CC_ChangeEPS	0x00000124	
TPM_CC_ChangePPS	0x00000125	
TPM_CC_Clear	0x00000126	
TPM_CC_ClearControl	0x00000127	
TPM_CC_ClockSet	0x00000128	
TPM_CC_HierarchyChangeAuth	0x00000129	
TPM_CC_NV_DefineSpace	0x0000012A	
TPM_CC_PCR_Allocate	0x0000012B	
TPM_CC_PCR_SetAuthPolicy	0x0000012C	
TPM_CC_PP_Commands	0x0000012D	
TPM_CC_SetPrimaryPolicy	0x0000012E	
TPM_CC_FieldUpgradeStart	0x0000012F	
TPM_CC_ClockRateAdjust	0x00000130	
TPM_CC_CreatePrimary	0x00000131	
TPM_CC_NV_GlobalWriteLock	0x00000132	
TPM_CC_GetCommandAuditDigest	0x00000133	
TPM_CC_NV_Increment	0x00000134	
TPM_CC_NV_SetBits	0x00000135	
TPM_CC_NV_Extend	0x00000136	
TPM_CC_NV_Write	0x00000137	
TPM_CC_NV_WriteLock	0x00000138	
TPM_CC_DictionaryAttackLockReset	0x00000139	
TPM_CC_DictionaryAttackParameters	0x0000013A	
TPM_CC_NV_ChangeAuth	0x0000013B	

Name	Command Code	Comments
TPM_CC_PCR_Event	0x0000013C	PCR
TPM_CC_PCR_Reset	0x0000013D	PCR
TPM_CC_SequenceComplete	0x0000013E	
TPM_CC_SetAlgorithmSet	0x0000013F	
TPM_CC_SetCommandCodeAuditStatus	0x00000140	
TPM_CC_FieldUpgradeData	0x00000141	
TPM_CC_IncrementalSelfTest	0x00000142	
TPM_CC_SelfTest	0x00000143	
TPM_CC_Startup	0x00000144	
TPM_CC_Shutdown	0x00000145	
TPM_CC_StirRandom	0x00000146	
TPM_CC_ActivateCredential	0x00000147	
TPM_CC_Certify	0x00000148	
TPM_CC_PolicyNV	0x00000149	Policy
TPM_CC_CertifyCreation	0x0000014A	
TPM_CC_Duplicate	0x0000014B	
TPM_CC_GetTime	0x0000014C	
TPM_CC_GetSessionAuditDigest	0x0000014D	
TPM_CC_NV_Read	0x0000014E	
TPM_CC_NV_ReadLock	0x0000014F	
TPM_CC_ObjectChangeAuth	0x00000150	
TPM_CC_PolicySecret	0x00000151	Policy
TPM_CC_Rewrap	0x00000152	
TPM_CC_Create	0x00000153	
TPM_CC_ECDH_ZGen	0x00000154	
TPM_CC_HMAC	0x00000155	
TPM_CC_Import	0x00000156	
TPM_CC_Load	0x00000157	
TPM_CC_Quote	0x00000158	
TPM_CC_RSA_Decrypt	0x00000159	
TPM_CC_HMAC_Start	0x0000015B	
TPM_CC_SequenceUpdate	0x0000015C	
TPM_CC_Sign	0x0000015D	
TPM_CC_Unseal	0x0000015E	

Name	Command Code	Comments
TPM_CC_PolicySigned	0x00000160	Policy
TPM_CC_ContextLoad	0x00000161	Context
TPM_CC_ContextSave	0x00000162	Context
TPM_CC_ECDH_KeyGen	0x00000163	
TPM_CC_EncryptDecrypt	0x00000164	
TPM_CC_FlushContext	0x00000165	Context
TPM_CC_LoadExternal	0x00000167	
TPM_CC_MakeCredential	0x00000168	
TPM_CC_NV_ReadPublic	0x00000169	NV
TPM_CC_PolicyAuthorize	0x0000016A	Policy
TPM_CC_PolicyAuthValue	0x0000016B	Policy
TPM_CC_PolicyCommandCode	0x0000016C	Policy
TPM_CC_PolicyCounterTimer	0x0000016D	Policy
TPM_CC_PolicyCpHash	0x0000016E	Policy
TPM_CC_PolicyLocality	0x0000016F	Policy
TPM_CC_PolicyNameHash	0x00000170	Policy
TPM_CC_PolicyOR	0x00000171	Policy
TPM_CC_PolicyTicket	0x00000172	Policy
TPM_CC_ReadPublic	0x00000173	
TPM_CC_RSA_Encrypt	0x00000174	
TPM_CC_StartAuthSession	0x00000176	
TPM_CC_VerifySignature	0x00000177	
TPM_CC_ECC_Parameters	0x00000178	
TPM_CC_FirmwareRead	0x00000179	
TPM_CC_GetCapability	0x0000017A	
TPM_CC_GetRandom	0x0000017B	
TPM_CC_GetTestResult	0x0000017C	
TPM_CC_Hash	0x0000017D	
TPM_CC_PCR_Read	0x0000017E	PCR
TPM_CC_PolicyPCR	0x0000017F	Policy
TPM_CC_PolicyRestart	0x00000180	
TPM_CC_ReadClock	0x00000181	
TPM_CC_PCR_Extend	0x00000182	
TPM_CC_PCR_SetAuthValue	0x00000183	
TPM_CC_NV_Certify	0x00000184	

Name	Command Code	Comments
TPM_CC_EventSequenceComplete	0x00000185	
TPM_CC_HashSequenceStart	0x00000186	
TPM_CC_PolicyPhysicalPresence	0x00000187	Policy
TPM_CC_PolicyDuplicationSelect	0x00000188	Policy
TPM_CC_PolicyGetDigest	0x00000189	Policy
TPM_CC_TestParms	0x0000018A	
TPM_CC_Commit	0x0000018B	
TPM_CC_PolicyPassword	0x0000018C	Policy
TPM_CC_ZGen_2Phase	0x0000018D	
TPM_CC_EC_Ephemeral	0x0000018E	
TPM_CC_PolicyNvWritten	0x0000018F	Policy
TPM_CC_PolicyTemplate	0x00000190	Policy
TPM_CC_CreateLoaded	0x00000191	
TPM_CC_PolicyAuthorizeNV	0x00000192	Policy
TPM_CC_EncryptDecrypt2	0x00000193	
TPM_CC_LAST	0x00000193	Compile variable. May increase based on implementation.
CC_VEND	0x20000000	
TPM_CC_Vendor_TCG_Test	CC_VEND+0x0000	Used for testing of command dispatch
#TPM_RC_COMMAND_CODE		

6.6 TPM_RC (Response Codes)

6.6.1 Description

Each return from the TPM has a 32-bit response code. The TPM will always set the upper 20 bits (31:12) of the response code to 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 this specification will use a tag of either TPM_ST_NO_SESSIONS or TPM_ST_SESSIONS. Error responses will use a tag value of TPM_ST_NO_SESSIONS and the response code will be as defined in this specification. Commands that use tags defined in the TPM 1.2 specification will use TPM_TAG_RSP_COMMAND in an error and a response code defined in TPM 1.2.

If the tag of the command is not a recognized command tag, the TPM error response will differ depending on TPM 1.2 compatibility. If the TPM supports 1.2 compatibility, the TPM shall return a tag of TPM_TAG_RSP_COMMAND and an appropriate TPM 1.2 response code (TPM_BADTAG = 00 00 00 $1E_{16}$). If the TPM does not have compatibility with TPM 1.2, the TPM shall return TPM_ST_NO_SESSION and a response code of TPM_RC_TAG.

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

6.6.2 Response Code Formats

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

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

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

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

bit	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 specification and is returned when the response tag is TPM_ST_NO_SESSIONS. **CLEAR (0)**: The error number is defined by a previous TPM specification. The error number is returned when the response tag is TPM_TAG_RSP_COMMAND. NOTE In any error number returned by a TPM, the F (bit 7) and V (bit 8) attributes shall be CLEAR when the response tag is TPM_TAG_RSP_COMMAND value used in TPM 1.2.
09	Reserved	shall be zero.
10	T	TCG/Vendor indicator **SET (1):** The response code is defined by the TPM vendor. **CLEAR (0):** The response code is defined by the TCG (a value in this specification). NOTE This attribute does not indicate a vendor-specific code unless the *F* attribute (bit[07]) is CLEAR.
11	S	severity **SET (1)**: The response code is a warning and the command was not necessarily in error. This command indicates that the TPM is busy or that the resources of the TPM have to be adjusted in order to allow the command to execute. **CLEAR (0)**: The response code indicates that the command had an error that would prevent it from running.

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

bit	11	10	09	08	07	06	05	04	03	02	01	00
	N				1	P	E					

Figure 3 — Format-One Response Codes

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

NOTE 1 In the reference implementation, for a RC_FMT1 response code, a constant of the form RC_Command_parameterName is the one based parameter number (TPM_RC_n) plus TPM_RC_P.

Example RC_Startup_startupType is the first parameter, TPM_RC_1 (0x100) plus TPM_RC_P (0x40) or 0x140. TPM_RC_VALUE (RC_FMT1 (0x080) + 0x004) + RC_Startup_startupType is thus 0x080 + 0x004 + 0x140 = 0x1c4.

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

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

The field definitions are:

Table 14 — Format-One Response Codes

Bit	Name	Definition
05:00	E	the error number The error number is independent of the other settings.
06	P	**SET (1):** The error is associated with a parameter. **CLEAR (0):** The error is associated with a handle or a session.
07	F	the response code format selector This field shall be SET for the format in this table.
11:08	N	the number of the handle, session, or parameter in error. The number is one based. See TPM_RC_1 through TPM_RC_F. If P is SET, then this field is the parameter in error. If P is CLEAR, then this field indicates the handle or session in error. Handles use values of N between 0000_2 and 0111_2. Sessions use values between 1000_2 and 1111_2. NOTE Bit 11 distinguishes between handles and sessions. Bits 10:8 000_2 indicate that the number is unspecified.

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

Table 15 — Response Code Groupings

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

6.6.3 TPM_RC Values

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

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

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

Name	Value	Description
TPM_RC_SUCCESS	0x000	
TPM_RC_BAD_TAG	0x01E	defined for compatibility with TPM 1.2
RC_VER1	0x100	set for all format 0 response codes
TPM_RC_INITIALIZE	RC_VER1 + 0x000	TPM not initialized by TPM2_Startup or already initialized
TPM_RC_FAILURE	RC_VER1 + 0x001	commands not being accepted because of a TPM failure NOTE This may be returned by TPM2_GetTestResult() as the *testResult* parameter.
TPM_RC_SEQUENCE	RC_VER1 + 0x003	improper use of a sequence handle
TPM_RC_PRIVATE	RC_VER1 + 0x00B	not currently used
TPM_RC_HMAC	RC_VER1 + 0x019	not currently used
TPM_RC_DISABLED	RC_VER1 + 0x020	the command is disabled
TPM_RC_EXCLUSIVE	RC_VER1 + 0x021	command failed because audit sequence required exclusivity
TPM_RC_AUTH_TYPE	RC_VER1 + 0x024	authorization handle is not correct for command
TPM_RC_AUTH_MISSING	RC_VER1 + 0x025	command requires an authorization session for handle and it is not present.
TPM_RC_POLICY	RC_VER1 + 0x026	policy failure in math operation or an invalid authPolicy value
TPM_RC_PCR	RC_VER1 + 0x027	PCR check fail
TPM_RC_PCR_CHANGED	RC_VER1 + 0x028	PCR have changed since checked.
TPM_RC_UPGRADE	RC_VER1 + 0x02D	for all commands other than TPM2_FieldUpgradeData(), this code indicates that the TPM is in field upgrade mode; for TPM2_FieldUpgradeData(), this code indicates that the TPM is not in field upgrade mode
TPM_RC_TOO_MANY_CONTEXTS	RC_VER1 + 0x02E	context ID counter is at maximum.
TPM_RC_AUTH_UNAVAILABLE	RC_VER1 + 0x02F	authValue or authPolicy is not available for selected entity.
TPM_RC_REBOOT	RC_VER1 + 0x030	a _TPM_Init and Startup(CLEAR) is required before the TPM can resume operation.
TPM_RC_UNBALANCED	RC_VER1 + 0x031	the protection algorithms (hash and symmetric) are not reasonably balanced. The digest size of the hash must be larger than the key size of the symmetric algorithm.

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

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 or nonce value mismatch
TPM_RC_PP	RC_FMT1 + 0x010	authorization requires assertion of PP
TPM_RC_SCHEME	RC_FMT1 + 0x012	unsupported or incompatible scheme
TPM_RC_SIZE	RC_FMT1 + 0x015	structure is the wrong size
TPM_RC_SYMMETRIC	RC_FMT1 + 0x016	unsupported symmetric algorithm or key size, or not appropriate for instance
TPM_RC_TAG	RC_FMT1 + 0x017	incorrect structure tag
TPM_RC_SELECTOR	RC_FMT1 + 0x018	union selector is incorrect
TPM_RC_INSUFFICIENT	RC_FMT1 + 0x01A	the TPM was unable to unmarshal a value because there were not enough octets in the input buffer
TPM_RC_SIGNATURE	RC_FMT1 + 0x01B	the signature is not valid
TPM_RC_KEY	RC_FMT1 + 0x01C	key fields are not compatible with the selected use
TPM_RC_POLICY_FAIL	RC_FMT1 + 0x01D	a policy check failed
TPM_RC_INTEGRITY	RC_FMT1 + 0x01F	integrity check failed
TPM_RC_TICKET	RC_FMT1 + 0x020	invalid ticket
TPM_RC_RESERVED_BITS	RC_FMT1 + 0x021	reserved bits not set to zero as required
TPM_RC_BAD_AUTH	RC_FMT1 + 0x022	authorization failure without DA implications
TPM_RC_EXPIRED	RC_FMT1 + 0x023	the policy has expired
TPM_RC_POLICY_CC	RC_FMT1 + 0x024	the *commandCode* in the policy is not the *commandCode* of the command or the command code in a policy command references a command that is not implemented
TPM_RC_BINDING	RC_FMT1 + 0x025	public and sensitive portions of an object are not cryptographically bound
TPM_RC_CURVE	RC_FMT1 + 0x026	curve not supported
TPM_RC_ECC_POINT	RC_FMT1 + 0x027	point is not on the required curve.
		New Subsection
RC_WARN	0x900	set for warning response codes

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 NOTE 1 This cannot occur on the reference implementation. NOTE 2 There is no reason why an implementation would implement a design that would deplete handle space. Platform specifications are encouraged to forbid it.
TPM_RC_LOCALITY	RC_WARN + 0x007	bad locality
TPM_RC_YIELDED	RC_WARN + 0x008	the TPM has suspended operation on the command; forward progress was made and the command may be retried See TPM 2.0 Part 1, “Multi-tasking.” NOTE This cannot occur on the reference implementation.
TPM_RC_CANCELED	RC_WARN + 0x009	the command was canceled
TPM_RC_TESTING	RC_WARN + 0x00A	TPM is performing self-tests
TPM_RC_REFERENCE_H0	RC_WARN + 0x010	the 1st handle in the handle area references a transient object or session that is not loaded
TPM_RC_REFERENCE_H1	RC_WARN + 0x011	the 2nd handle in the handle area references a transient object or session that is not loaded
TPM_RC_REFERENCE_H2	RC_WARN + 0x012	the 3rd handle in the handle area references a transient object or session that is not loaded
TPM_RC_REFERENCE_H3	RC_WARN + 0x013	the 4th handle in the handle area references a transient object or session that is not loaded
TPM_RC_REFERENCE_H4	RC_WARN + 0x014	the 5th handle in the handle area references a transient object or session that is not loaded
TPM_RC_REFERENCE_H5	RC_WARN + 0x015	the 6th handle in the handle area references a transient object or session that is not loaded
TPM_RC_REFERENCE_H6	RC_WARN + 0x016	the 7th handle in the handle area references a transient object or session that is not loaded
TPM_RC_REFERENCE_S0	RC_WARN + 0x018	the 1st authorization session handle references a session that is not loaded
TPM_RC_REFERENCE_S1	RC_WARN + 0x019	the 2nd authorization session handle references a session that is not loaded
TPM_RC_REFERENCE_S2	RC_WARN + 0x01A	the 3rd authorization session handle references a session that is not loaded
TPM_RC_REFERENCE_S3	RC_WARN + 0x01B	the 4th authorization session handle references a session that is not loaded
TPM_RC_REFERENCE_S4	RC_WARN + 0x01C	the 5th session handle references a session that is not loaded

Name	Value	Description
TPM_RC_REFERENCE_S5	RC_WARN + 0x01D	the 6th session handle references a session that is not loaded
TPM_RC_REFERENCE_S6	RC_WARN + 0x01E	the 7th authorization session handle references a session that is not loaded
TPM_RC_NV_RATE	RC_WARN + 0x020	the TPM is rate-limiting accesses to prevent wearout of NV
TPM_RC_LOCKOUT	RC_WARN + 0x021	authorizations for objects subject to DA protection are not allowed at this time because the TPM is in DA lockout mode
TPM_RC_RETRY	RC_WARN + 0x022	the TPM was not able to start the command
TPM_RC_NV_UNAVAILABLE	RC_WARN + 0x023	the command may require writing of NV and NV is not current accessible
TPM_RC_NOT_USED	RC_WARN + 0x7F	this value is reserved and shall not be returned by the TPM
		Additional Defines
TPM_RC_H	0x000	add to a handle-related error
TPM_RC_P	0x040	add to a parameter-related error
TPM_RC_S	0x800	add to a session-related error
TPM_RC_1	0x100	add to a parameter-, handle-, or session-related error
TPM_RC_2	0x200	add to a parameter-, handle-, or session-related error
TPM_RC_3	0x300	add to a parameter-, handle-, or session-related error
TPM_RC_4	0x400	add to a parameter-, handle-, or session-related error
TPM_RC_5	0x500	add to a parameter-, handle-, or session-related error
TPM_RC_6	0x600	add to a parameter-, handle-, or session-related error
TPM_RC_7	0x700	add to a parameter-, handle-, or session-related error
TPM_RC_8	0x800	add to a parameter-related error
TPM_RC_9	0x900	add to a parameter-related error
TPM_RC_A	0xA00	add to a parameter-related error
TPM_RC_B	0xB00	add to a parameter-related error
TPM_RC_C	0xC00	add to a parameter-related error
TPM_RC_D	0xD00	add to a parameter-related error
TPM_RC_E	0xE00	add to a parameter-related error
TPM_RC_F	0xF00	add to a parameter-related error
TPM_RC_N_MASK	0xF00	number mask

6.7 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		

6.8 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

6.9 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 earlier versions of this specification. This value is used when the TPM is compatible with a previous TPM specification and the TPM cannot determine which family of response code to return because the command tag is not valid.

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

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

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

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

Name	Value	Comments
TPM_ST_RSP_COMMAND	0x00C4	*tag* value for a response; used when there is an error in the tag. This is also the value returned from a TPM 1.2 when an error occurs. This value is used in this specification because an error in the command tag may prevent determination of the family. When this tag is used in the response, the response code will be TPM_RC_BAD_TAG (0 $1E_{16}$), which has the same numeric value as the TPM 1.2 response code for TPM_BADTAG. NOTE In a previously published version of this specification, TPM_RC_BAD_TAG was incorrectly assigned a value of 0x030 instead of 30 (0x01e). Some implementations my return the old value instead of the new value.
TPM_ST_NULL	0X8000	no structure type specified
TPM_ST_NO_SESSIONS	0x8001	*tag* value for a command/response for a command defined in this specification; indicating that the command/response has no attached sessions and no *authorizationSize*/*parameterSize* value is present If the *responseCode* from the TPM is not TPM_RC_SUCCESS, then the response tag shall have this value.
TPM_ST_SESSIONS	0x8002	*tag* value for a command/response for a command defined in this specification; indicating that the command/response has one or more attached sessions and the *authorizationSize*/*parameterSize* field is present

Name	Value	Comments
reserved	0x8003	When used between application software and the TPM resource manager, this tag indicates that the command has no sessions and the handles are using the Name format rather than the 32-bit handle format. NOTE 1 The response to application software will have a *tag* of TPM_ST_NO_SESSIONS. Between the TRM and TPM, this tag would occur in a response from a TPM that overlaps the *tag* parameter of a request with the *tag* parameter of a response, when the response has no associated sessions. NOTE 2 This tag is not used by all TPM or TRM implementations.
reserved	0x8004	When used between application software and the TPM resource manager, this tag indicates that the command has sessions and the handles are using the Name format rather than the 32-bit handle format. NOTE 1 If the command completes successfully, the response to application software will have a *tag* of TPM_ST_SESSIONS. Between the TRM and TPM, would occur in a response from a TPM that overlaps the *tag* parameter of a request with the *tag* parameter of a response, when the response has authorization sessions. NOTE 2 This tag is not used by all TPM or TRM implementations.
TPM_ST_ATTEST_NV	0x8014	tag for an attestation structure
TPM_ST_ATTEST_COMMAND_AUDIT	0x8015	tag for an attestation structure
TPM_ST_ATTEST_SESSION_AUDIT	0x8016	tag for an attestation structure
TPM_ST_ATTEST_CERTIFY	0x8017	tag for an attestation structure
TPM_ST_ATTEST_QUOTE	0x8018	tag for an attestation structure
TPM_ST_ATTEST_TIME	0x8019	tag for an attestation structure
TPM_ST_ATTEST_CREATION	0x801A	tag for an attestation structure
reserved	0x801B	do not use NOTE This was previously assigned to TPM_ST_ATTEST_NV. The tag is changed because the structure has changed
TPM_ST_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

6.10 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

6.11 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

6.12 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_AUTH_POLICIES	0x00000009		TPML_TAGGED_POLICY
TPM_CAP_LAST	0x00000009		
TPM_CAP_VENDOR_PROPERTY	0x00000100	manufacturer specific	manufacturer-specific values
#TPM_RC_VALUE			

NOTES:
(1) The TPM_ALG_ID or TPM_ECC_CURVE is cast to a UINT32

6.13 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. NOTE The first group with any properties is group 1 (PT_GROUP * 1). Group 0 is reserved.
PT_FIXED	PT_GROUP * 1	the group of fixed properties returned as TPMS_TAGGED_PROPERTY The values in this group are only changed due to a firmware change in the TPM.
TPM_PT_FAMILY_INDICATOR	PT_FIXED + 0	a 4-octet character string containing the TPM Family value (TPM_SPEC_FAMILY)
TPM_PT_LEVEL	PT_FIXED + 1	the level of the specification NOTE 1 For this specification, the level is zero. NOTE 2 The level is on the title page of the specification.
TPM_PT_REVISION	PT_FIXED + 2	the specification Revision times 100 EXAMPLE Revision 01.01 would have a value of 101. NOTE The Revision value is on the title page of the specification.
TPM_PT_DAY_OF_YEAR	PT_FIXED + 3	the specification day of year using TCG calendar EXAMPLE November 15, 2010, has a day of year value of 319 (00 00 01 $3F_{16}$). NOTE The specification date is on the title page of the specification.
TPM_PT_YEAR	PT_FIXED + 4	the specification year using the CE EXAMPLE The year 2010 has a value of 00 00 07 DA_{16}. NOTE The specification date is on the title page of the specification.
TPM_PT_MANUFACTURER	PT_FIXED + 5	the vendor ID unique to each TPM manufacturer
TPM_PT_VENDOR_STRING_1	PT_FIXED + 6	the first four characters of the vendor ID string NOTE When the vendor string is fewer than 16 octets, the additional property values do not have to be present. A vendor string of 4 octets can be represented in one 32-bit value and no null terminating character is required.
TPM_PT_VENDOR_STRING_2	PT_FIXED + 7	the second four characters of the vendor ID string
TPM_PT_VENDOR_STRING_3	PT_FIXED + 8	the third four characters of the vendor ID string
TPM_PT_VENDOR_STRING_4	PT_FIXED + 9	the fourth four characters of the vendor ID sting
TPM_PT_VENDOR_TPM_TYPE	PT_FIXED + 10	vendor-defined value indicating the TPM model
TPM_PT_FIRMWARE_VERSION_1	PT_FIXED + 11	the most-significant 32 bits of a TPM vendor-specific value indicating the version number of the firmware. See 10.12.2 and 10.12.8.

Capability Name	Value	Comments
TPM_PT_FIRMWARE_VERSION_2	PT_FIXED + 12	the least-significant 32 bits of a TPM vendor-specific value indicating the version number of the firmware. See 10.12.2 and 10.12.8.
TPM_PT_INPUT_BUFFER	PT_FIXED + 13	the maximum size of a parameter (typically, a TPM2B_MAX_BUFFER)
TPM_PT_HR_TRANSIENT_MIN	PT_FIXED + 14	the minimum number of transient objects that can be held in TPM RAM NOTE This minimum shall be no less than the minimum value required by the platform-specific specification to which the TPM is built.
TPM_PT_HR_PERSISTENT_MIN	PT_FIXED + 15	the minimum number of persistent objects that can be held in TPM NV memory NOTE This minimum shall be no less than the minimum value required by the platform-specific specification to which the TPM is built.
TPM_PT_HR_LOADED_MIN	PT_FIXED + 16	the minimum number of authorization sessions that can be held in TPM RAM NOTE This minimum shall be no less than the minimum value required by the platform-specific specification to which the TPM is built.
TPM_PT_ACTIVE_SESSIONS_MAX	PT_FIXED + 17	the number of authorization sessions that may be active at a time A session is active when it has a context associated with its handle. The context may either be in TPM RAM or be context saved. NOTE This value shall be no less than the minimum value required by the platform-specific specification to which the TPM is built.
TPM_PT_PCR_COUNT	PT_FIXED + 18	the number of PCR implemented NOTE This number is determined by the defined attributes, not the number of PCR that are populated.
TPM_PT_PCR_SELECT_MIN	PT_FIXED + 19	the minimum number of octets in a TPMS_PCR_SELECT.*sizeOfSelect* NOTE This value is not determined by the number of PCR implemented but by the number of PCR required by the platform-specific specification with which the TPM is compliant or by the implementer if not adhering to a platform-specific specification.
TPM_PT_CONTEXT_GAP_MAX	PT_FIXED + 20	the maximum allowed difference (unsigned) between the *contextID* values of two saved session contexts This value shall be 2^n-1, where n is at least 16.
	PT_FIXED + 21	skipped
TPM_PT_NV_COUNTERS_MAX	PT_FIXED + 22	the maximum number of NV Indexes that are allowed to have the TPM_NT_COUNTER attribute NOTE It is allowed for this value to be larger than the number of NV Indexes that can be defined. This would be indicative of a TPM implementation that did not use different implementation technology for different NV Index types.
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

Capability Name	Value	Comments
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 $1 \leq N \leq 32$ NOTE 1 An "orderly counter" is an NV Index with an TPM_NT of TPM_NV_COUNTER and TPMA_NV_ORDERLY SET. NOTE 2 When the low-order bits of a counter equal this value, an NV write occurs on the next increment.
TPM_PT_MAX_COMMAND_SIZE	PT_FIXED + 30	the maximum value for *commandSize* in a command
TPM_PT_MAX_RESPONSE_SIZE	PT_FIXED + 31	the maximum value for *responseSize* in a response
TPM_PT_MAX_DIGEST	PT_FIXED + 32	the maximum size of a digest that can be produced by the TPM
TPM_PT_MAX_OBJECT_CONTEXT	PT_FIXED + 33	the maximum size of an object context that will be returned by TPM2_ContextSave
TPM_PT_MAX_SESSION_CONTEXT	PT_FIXED + 34	the maximum size of a session context that will be returned by TPM2_ContextSave
TPM_PT_PS_FAMILY_INDICATOR	PT_FIXED + 35	platform-specific family (a TPM_PS value)(see Table 25) NOTE The platform-specific values for the TPM_PT_PS parameters are in the relevant platform-specific specification. In the reference implementation, all of these values are 0.
TPM_PT_PS_LEVEL	PT_FIXED + 36	the level of the platform-specific specification
TPM_PT_PS_REVISION	PT_FIXED + 37	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
TPM_PT_VENDOR_COMMANDS	PT_FIXED + 43	number of vendor commands that are implemented
TPM_PT_NV_BUFFER_MAX	PT_FIXED + 44	the maximum data size in one NV write, NV read, or NV certify command
TPM_PT_MODES	PT_FIXED + 45	a TPMA_MODES value, indicating that the TPM is designed for these modes.
TPM_PT_MAX_CAP_BUFFER	PT_FIXED + 46	the maximum size of a TPMS_CAPABILITY_DATA structure returned in TPM2_GetCapability().
		Intentionally left empty

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

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

6.14 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. If there is no property that corresponds to the value of *property*, the next higher value is returned, if it exists.

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

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

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 This is an implementation-specific value. The value shown reflects the reference code implementation.

6.15 TPM_PS (Platform Specific)

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

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

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

Capability Name	Value	Comments
TPM_PS_MAIN	0x00000000	not platform specific
TPM_PS_PC	0x00000001	PC Client
TPM_PS_PDA	0x00000002	PDA (includes all mobile devices that are not specifically cell phones)
TPM_PS_CELL_PHONE	0x00000003	Cell Phone
TPM_PS_SERVER	0x00000004	Server WG
TPM_PS_PERIPHERAL	0x00000005	Peripheral WG
TPM_PS_TSS	0x00000006	TSS WG
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

7 Handles

7.1 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.

7.2 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_SAVED_SESSION	0x03	**Saved Authorization Session** – used only in the context of TPM2_GetCapability This type references saved authorization session contexts for which the TPM is maintaining tracking information.
TPM_HT_PERMANENT	0x40	**Permanent Values** – assigned by this specification in Table 28
TPM_HT_TRANSIENT	0x80	**Transient Objects** – assigned by the TPM when an object is loaded into transient-object memory or when a persistent object is converted to a transient object
TPM_HT_PERSISTENT	0x81	**Persistent Objects** – assigned by the TPM when a loaded transient object is made persistent

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.

7.3 Persistent Handle Sub-ranges

Persistent handles are assigned by the caller of TPM2_EvictControl(). Owner Authorization or Platform Authorization is required to authorize allocation of space for a persistent object. These entities are given separate ranges of persistent handles so that they do not have to allocate from a common range of handles.

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

The Owner is allocated persistent handles in the range of 81 00 00 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.

7.4 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[1]
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[1]
TPM_RH_TRANSPORT	0x40000003	R	not used[1]
TPM_RH_OPERATOR	0x40000004	R	not used[1]
TPM_RH_ADMIN	0x40000005	R	not used[1]
TPM_RH_EK	0x40000006	R	not used[1]
TPM_RH_NULL	0x40000007	K, A, P	a handle associated with the null hierarchy, an EmptyAuth *authValue*, and an Empty Policy *authPolicy*.
TPM_RH_UNASSIGNED	0x40000008	R	value reserved to the TPM to indicate a handle location that has not been initialized or assigned
TPM_RS_PW	0x40000009	S	authorization value used to indicate a password authorization session
TPM_RH_LOCKOUT	0x4000000A	A	references the authorization associated with the dictionary attack lockout reset
TPM_RH_ENDORSEMENT	0x4000000B	K, A, P	references the Endorsement Primary Seed (EPS), *endorsementAuth*, and *endorsementPolicy*
TPM_RH_PLATFORM	0x4000000C	K, A, P	references the Platform Primary Seed (PPS), *platformAuth*, and *platformPolicy*
TPM_RH_PLATFORM_NV	0x4000000D	C	for phEnableNV
TPM_RH_AUTH_00	0x40000010	A	Start of a range of authorization values that are vendor-specific. A TPM may support any of the values in this range as are needed for vendor-specific purposes. Disabled if ehEnable is CLEAR. NOTE “Any” includes “none”.
TPM_RH_AUTH_FF	0x4000010F	A	End of the range of vendor-specific authorization values.
TPM_RH_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 handle is only used in a TPM that is compatible with a previous version of this specification. It is not used in any command defined in this version of the specification.

7.5 TPM_HC (Handle Value Constants)

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

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

- HMAC_SESSION_FIRST—HMAC_SESSION_LAST,
- LOADED_SESSION_FIRST—LOADED_SESSION_LAST,
- POLICY_SESSION_FIRST—POLICY_SESSION_LAST,
- TRANSIENT_FIRST—TRANSIENT_LAST,
- ACTIVE_SESSION_FIRST—ACTIVE_SESSION_LAST,
- PCR_FIRST—PCR_LAST

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

- PERSISTENT_FIRST—PERSISTENT_LAST,
- PLATFORM_PERSISTENT—PLATFORM_PERSISTENT+0x007FFFFF,
- NV_INDEX_FIRST—NV_INDEX_LAST,
- PERMANENT_FIRST—PERMANENT_LAST

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

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

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

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

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

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

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

8 Attribute Structures

8.1 Description

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

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

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

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

8.2 TPMA_ALGORITHM

This structure defines the attributes of an algorithm.

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

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

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

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

Table 30 — Definition of (UINT32) TPMA_ALGORITHM Bits

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

8.3 TPMA_OBJECT (Object Attributes)

8.3.1 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.

8.3.2 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

Bit	Name	Definition
16	restricted	**SET (1):** Key usage is restricted to manipulate structures of known format; the parent of this key shall have *restricted* SET. **CLEAR (0):** Key usage is not restricted to use on special formats.
17	decrypt	**SET (1):** The private portion of the key may be used to decrypt. **CLEAR (0):** The private portion of the key may not be used to decrypt.
18	sign / encrypt	**SET (1):** For a symmetric cipher object, the private portion of the key may be used to encrypt. For other objects, the private portion of the key may be used to sign. **CLEAR (0):** The private portion of the key may not be used to sign or encrypt.
31:19	Reserved	shall be zero

8.3.3 Attribute Descriptions

8.3.3.1 Introduction

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

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

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

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

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

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

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

8.3.3.2 Bit[1] – *fixedTPM*

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

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

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

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

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

Import shall be CLEAR

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

8.3.3.3 Bit[2] – *stClear*

If this attribute is SET, then saved contexts of this object will be invalidated on TPM2_Startup(TPM_SU_CLEAR). If the attribute is CLEAR, then the TPM shall not invalidate the saved context if the TPM received TPM2_Shutdown(TPM_SU_STATE). If the saved state is valid when checked at the next TPM2_Startup(), then the TPM shall continue to be able to use the saved contexts.

Creation may be SET or CLEAR in template

Load may be SET or CLEAR

Import may be SET or CLEAR

External may be SET or CLEAR

8.3.3.4 Bit[4] – *fixedParent*

If this attribute is SET, the object's parent may not be changed. That is, this object may not be the object of a TPM2_Duplicate(). If this attribute is CLEAR, then this object may be the object of a TPM2_Duplicate().

Creation may be SET or CLEAR in template

Load may be SET or CLEAR

Import shall be CLEAR

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

8.3.3.5 Bit[5] – *sensitiveDataOrigin*

This attribute is SET for any key that was generated by TPM in TPM2_Create() or TPM2_CreatePrimary(). If CLEAR, it indicates that the sensitive part of the object (other than the *obfuscation* value) was provided by the caller.

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

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

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

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

Load may be SET or CLEAR

Import may be SET or CLEAR

External may be SET or CLEAR

8.3.3.6 Bit[6] – *userWithAuth*

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

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

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

NOTE 2 Regardless of the setting of this attribute, the *authValue* may be referenced in a policy session or used to provide the *bind* value in TPM2_StartAuthSession(). However, if *userWithAuth* is CLEAR, then the object may be used as the bind object in TPM2_StartAuthSession() but the session cannot be used to authorize actions on the object. If this were allowed, then the *userWithAuth* control could be circumvented simply by using the object as the bind object.

Creation may be SET or CLEAR in template

Load may be SET or CLEAR

Import may be SET or CLEAR

External may be SET or CLEAR

8.3.3.7 Bit[7] – *adminWithPolicy*

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

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

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

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

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

Creation may be SET or CLEAR in *template*

Load may be SET or CLEAR

Import may be SET or CLEAR

External may be SET or CLEAR

8.3.3.8 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

8.3.3.9 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* may not be SET if the object cannot be directly or indirectly duplicated.

If an object's parent has *fixedTPM* SET, and the object is duplicable (*fixedParent* == CLEAR), then *encryptedDuplication* may be SET or CLEAR in the object.

NOTE 3 This allows the object at the boundary between duplicable and non-duplicable objects to have either setting.

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

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

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

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

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

External may be SET or CLEAR.

8.3.3.10 Bit[16] – *restricted*

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

NOTE A key with this object CLEAR may not be a parent for another object.

Creation shall be CLEAR in *template* if neither sign nor decrypt is SET in *template*.

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

Import may be SET or CLEAR

External shall be CLEAR

8.3.3.11 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 should not be returned. The only type of data that may be returned after decryption is a Sealed Data Object (a *keyedHash* object with *decrypt* and *sign* CLEAR).

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

NOTE 2 A key with this attribute SET may be a parent for another object if *restricted* is SET and *sign* is CLEAR.

If *decrypt* is SET on an object with *type* set to TPM_ALG_KEYEDHASH, it indicates that the object is an XOR encryption key.

Creation may be SET or CLEAR in *template*

Load may be SET or CLEAR

Import may be SET or CLEAR

External may be SET or CLEAR

8.3.3.12 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 may not support both *sign* and *decrypt* being SET in the same key.

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

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

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

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

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

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

8.4 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.

If a session is not being used for authorization, at least one of decrypt, encrypt, or audit must be SET.

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

Bit	Name	Meaning
0	continueSession	**SET (1):** In a command, this setting indicates that the session is to remain active after successful completion of the command. In a response, it indicates that the session is still active. If SET in the command, this attribute shall be SET in the response. **CLEAR (0):** In a command, this setting indicates that the TPM should close the session and flush any related context when the command completes successfully. In a response, it indicates that the session is closed and the context is no longer active. This attribute has no meaning for a password authorization and the TPM will allow any setting of the attribute in the command and SET the attribute in the response. This attribute will only be CLEAR in one response for a logical session. If the attribute is CLEAR, the context associated with the session is no longer in use and the space is available. A session created after another session is ended may have the same handle but logically is not the same session. This attribute has no effect if the command does not complete successfully.
1	auditExclusive	**SET (1):** In a command, this setting indicates that the command should only be executed if the session is exclusive at the start of the command. In a response, it indicates that the session is exclusive. This setting is only allowed if the *audit* attribute is SET (TPM_RC_ATTRIBUTES). **CLEAR (0):** In a command, indicates that the session need not be exclusive at the start of the command. In a response, indicates that the session is not exclusive. In this revision, if *audit* is CLEAR, *auditExclusive* must be CLEAR in the command and will be CLEAR in the response. In a future, revision, this bit may have a different meaning if *audit* is CLEAR. See "Exclusive Audit Session" clause in TPM 2.0 Part 1.
2	auditReset	**SET (1):** In a command, this setting indicates that the audit digest of the session should be initialized and the exclusive status of the session SET. This setting is only allowed if the *audit* attribute is SET (TPM_RC_ATTRIBUTES). **CLEAR (0):** In a command, indicates that the audit digest should not be initialized. This bit is always CLEAR in a response. In this revision, if *audit* is CLEAR, *auditReset* must be clear in the command and will be CLEAR in the response. In a future, revision, this bit may have a different meaning if *audit* is CLEAR.
4:3	Reserved	shall be CLEAR

Bit	Name	Meaning
5	decrypt	**SET (1):** In a command, this setting indicates that the first parameter in the command is symmetrically encrypted using the parameter encryption scheme described in TPM 2.0 Part 1. The TPM will decrypt the parameter after performing any HMAC computations and before unmarshaling the parameter. In a response, the attribute is copied from the request but has no effect on the response. **CLEAR (0):** Session not used for encryption. For a password authorization, this attribute will be CLEAR in both the command and response. This attribute may 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_).
6	encrypt	**SET (1):** In a command, this setting indicates that the TPM should use this session to encrypt the first parameter in the response. In a response, it indicates that the attribute was set in the command and that the TPM used the session to encrypt the first parameter in the response using the parameter encryption scheme described in TPM 2.0 Part 1. **CLEAR (0):** Session not used for encryption. For a password authorization, this attribute will be CLEAR in both the command and response. This attribute may 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.

8.5 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

8.6 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):** o*wnerAuth* has not been changed since TPM2_Clear().
1	endorsementAuthSet	**SET (1):** TPM2_HierarchyChangeAuth() with *endorsementAuth* has been executed since the last TPM2_Clear(). **CLEAR (0):** endorsementAuth has not been changed since TPM2_Clear().
2	lockoutAuthSet	**SET (1):** TPM2_HierarchyChangeAuth() with *lockoutAuth* has been executed since the last TPM2_Clear(). **CLEAR (0):** *lockoutAuth* has not been changed since TPM2_Clear().
7:3	Reserved	
8	disableClear	**SET (1):** TPM2_Clear() is disabled. **CLEAR (0):** TPM2_Clear() is enabled. NOTE See "TPM2_ClearControl" in TPM 2.0 Part 3 for details on changing this attribute.
9	inLockout	**SET (1):** The TPM is in lockout, when *failedTries* is equal to *maxTries*.
10	tpmGeneratedEPS	**SET (1):** The EPS was created by the TPM. **CLEAR (0):** The EPS was created outside of the TPM using a manufacturer-specific process.
31:11	Reserved	

8.7 TPMA_STARTUP_CLEAR

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

phEnable is SET on any TPM2_Startup. *shEnable*, *ehEnable*, and *phEnableNV* are SET on TPM Reset or TPM_Restart and preserved by TPM Resume.

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

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

Bit	Parameter	Description
0	phEnable	**SET (1):** The platform hierarchy is enabled and *platformAuth* or *platformPolicy* may be used for authorization. **CLEAR (0):** *platformAuth* and *platformPolicy* may not be used for authorizations, and objects in the platform hierarchy, including persistent objects, cannot be used. NOTE See "TPM2_HierarchyControl" in TPM 2.0 Part 3 for details on changing this attribute.
1	shEnable	**SET (1):** The Storage hierarchy is enabled and *ownerAuth* or *ownerPolicy* may be used for authorization. NV indices defined using owner authorization are accessible. **CLEAR (0):** *ownerAuth* and *ownerPolicy* may not be used for authorizations, and objects in the Storage hierarchy, persistent objects, and NV indices defined using owner authorization cannot be used. NOTE See "TPM2_HierarchyControl" in TPM 2.0 Part 3 for details on changing this attribute.
2	ehEnable	**SET (1):** The EPS hierarchy is enabled and Endorsement 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. NOTE See "TPM2_HierarchyControl" in TPM 2.0 Part 3 for details on changing this attribute.
3	phEnableNV	**SET (1):** NV indices that have TPMA_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. NOTE See "TPM2_HierarchyControl" in TPM 2.0 Part 3 for details on changing this attribute. NOTE read refers to these commands: TPM2_NV_Read, TPM2_NV_ReadPublic, TPM_NV_Certify, TPM2_PolicyNV write refers to these commands: TPM2_NV_Write, TPM2_NV_Increment, TPM2_NV_Extend, TPM2_NV_SetBits NOTE The TPM must query the index TPMA_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.
30:4	Reserved	shall be zero

Bit	Parameter	Description
31	orderly	**SET (1):** The TPM received a TPM2_Shutdown() and a matching TPM2_Startup(). **CLEAR (0):** TPM2_Startup(TPM_SU_CLEAR) was not preceded by a TPM2_Shutdown() of any type. NOTE A shutdown is orderly if the TPM receives a TPM2_Shutdown() of any type followed by a TPM2_Startup() of any type. However, the TPM will return an error if TPM2_Startup(TPM_SU_STATE) was not preceded by TPM2_Shutdown(TPM_SU_STATE).

8.8 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

8.9 TPMA_CC (Command Code Attributes)

8.9.1 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.

8.9.2 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 a version of this specification
31:30	Res	allocated for software; shall be zero

8.9.3 Field Descriptions

8.9.3.1 Bits[15:0] – *commandIndex*

This is the command index of the command in the set of commands. The two sets are defined by the *V* attribute. If *V* is zero, then the *commandIndex* shall be in the set of commands defined in a version of this specification. If *V* is one, then the meaning of *commandIndex* is as determined by the TPM vendor.

8.9.3.2 Bit[22] – *nv*

If this attribute is SET, then the TPM may perform an NV write as part of the command actions. This write is independent of any write that may occur as a result of dictionary attack protection. If this attribute is CLEAR, then the TPM shall not perform an NV write as part of the command actions.

8.9.3.3 Bit[23] – *extensive*

If this attribute is SET, then the TPM may flush many transient objects as a side effect of this command. In TPM 2.0 Part 3, a command that has this attribute is indicated by using a "{E}" decoration in the "Description" column of the *commandCode* parameter.

EXAMPLE See "TPM2_Clear" in TPM 2.0 Part 3.

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

8.9.3.4 Bit[24] – *flushed*

If this attribute is SET, then the TPM will flush transient objects as a side effect of this command. Any transient objects listed in the handle area of the command will be flushed from TPM memory. Handles associated with persistent objects, sessions, PCR, or other fixed TPM resources are not flushed.

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

NOTE The "{F}" decoration may be combined with other decorations such as "{NV}" in which case the decoration would be "{NV F}."

If this attribute is SET for a command, and the handle of the command is associated with a hierarchy (TPM_RH_PLATFORM, TPM_RH_OWNER, or TPM_RH_ENDORSEMENT), all loaded objects in the indicated hierarchy are flushed.

The TRM is expected to know the behaviour of TPM2_ContextSave(), and sessions are flushed when context saved, but objects are not. The *flushed* attribute for that command shall be CLEAR.

In TPM 2.0 Part 3, a command that has this attribute is indicated by using a "{F}" decoration in the "Description" column of the *commandCode* parameter.

EXAMPLE See "TPM2_SequenceComplete" in TPM 2.0 Part 3."

8.9.3.5 Bits[27:25] – *cHandles*

This field indicates the number of handles in the handle area of the command. This number allows the TRM to enumerate the handles in the handle area and find the position of the authorizations (if any).

8.9.3.6 Bit[28] – *rHandle*

If this attribute is SET, then the response to this command has a handle area. This area will contain no more than one handle. This field is necessary to allow the TRM to locate the *parameterSize* field in the response, which is then used to locate the authorizations.

NOTE The TRM is expected to "virtualize" the handle value for any returned handle.

A TPM command is only allowed to have one handle in the response handle area.

8.9.3.7 Bit[29] – V

When this attribute is SET, it indicates that the command operation is defined by the TPM vendor. When CLEAR, it indicates that the command is defined by a version of this specification.

8.9.3.8 Bits[31:30] – Res

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

8.10 TPMA_MODES

This structure of this attribute is used to report that the TPM is designed for these modes. This structure may be read using TPM2_GetCapability(*capability* = TPM_CAP_TPM_PROPERTIES, *property* = TPM_PT_MODES).

NOTE: To determine the certification status of a TPM with the FIPS_140_2 attribute SET, consult the NIST Module Validation List at http://csrc.nist.gov/groups/STM/cmvp/validation.html.

Table 38 — Definition of (UINT32) TPMA_MODES Bits <Out>

Bit	Name	Definition
0	FIPS_140_2	**SET (1):** indicates that the TPM is designed to comply with all of the FIPS 140-2 requirements at Level 1 or higher.
31:1	Reserved	shall be zero

9 Interface Types

9.1 Introduction

Clause 9 contains definitions for interface types. An interface type is type checked when it is unmarshaled. These types are based on an underlying type that is indicated in the table title by the value in parentheses. When an interface type is used, the base type is unmarshaled and then checked to see if it has one of the allowed values.

9.2 TPMI_YES_NO

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 39 — Definition of (BYTE) TPMI_YES_NO Type

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

9.3 TPMI_DH_OBJECT

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

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

Table 40 — Definition of (TPM_HANDLE) TPMI_DH_OBJECT Type

Values	Comments
{TRANSIENT_FIRST:TRANSIENT_LAST}	allowed range for transient objects
{PERSISTENT_FIRST:PERSISTENT_LAST}	allowed range for persistent objects
+TPM_RH_NULL	the conditional value
#TPM_RC_VALUE	

9.4 TPMI_DH_PARENT

The TPMI_DH_PARENT interface type is a handle that references an object that can be the parent of another object. The handles in this set may refer to either transient or persistent object or to Primary Seeds.

Table 41 — Definition of (TPM_HANDLE) TPMI_DH_PARENT Type

Values	Comments
{TRANSIENT_FIRST:TRANSIENT_LAST}	allowed range for transient objects
{PERSISTENT_FIRST:PERSISTENT_LAST}	allowed range for persistent objects
TPM_RH_OWNER	Storage hierarchy
TPM_RH_PLATFORM	Platform hierarchy
TPM_RH_ENDORSEMENT	Endorsement hierarchy
+TPM_RH_NULL	no hierarchy
#TPM_RC_VALUE	

9.5 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 42 — Definition of (TPM_HANDLE) TPMI_DH_PERSISTENT Type

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

9.6 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 43 — 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	

9.7 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 previous versions of TPM specifications.

Table 44 — Definition of (TPM_HANDLE) TPMI_DH_PCR Type <IN>

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

9.8 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 45 — 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

9.9 TPMI_SH_HMAC

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

Table 46 — 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

9.10 TPMI_SH_POLICY

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

Table 47 — 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

9.11 TPMI_DH_CONTEXT

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

Table 48 — Definition of (TPM_HANDLE) TPMI_DH_CONTEXT Type

Values	Comments
{HMAC_SESSION_FIRST : HMAC_SESSION_LAST}	
{POLICY_SESSION_FIRST:POLICY_SESSION_LAST}	
{TRANSIENT_FIRST:TRANSIENT_LAST}	
#TPM_RC_VALUE	

9.12 TPMI_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 49 — 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

9.13 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 50 — 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

9.14 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 51 — 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

9.15 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 52 — 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

9.16 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 53 — 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

9.17 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 54 — 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

9.18 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 55 — 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

9.19 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 56 — 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

9.20 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 57 — 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

9.21 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 58 — 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

9.22 TPMI_RH_NV_INDEX

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

Table 59 — 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

9.23 TPMI_ALG_HASH

A TPMI_ALG_HASH is an interface type of all the hash algorithms implemented on a specific TPM. The selector in Table 60 indicates all of the hash algorithms that have an algorithm ID assigned by the TCG and does not indicate the algorithms that will be accepted by a TPM.

NOTE When implemented, each of the algorithm entries is delimited by #ifdef and #endif so that, if the algorithm is not implemented in a specific TPM, that algorithm is not included in the interface type.

Table 60 — Definition of (TPM_ALG_ID) TPMI_ALG_HASH Type

Values	Comments
TPM_ALG_!ALG.H	all hash algorithms defined by the TCG
+TPM_ALG_NULL	
#TPM_RC_HASH	

9.24 TPMI_ALG_ASYM (Asymmetric Algorithms)

A TPMI_ALG_ASYM is an interface type of all the asymmetric algorithms implemented on a specific TPM. Table 61 lists each of the asymmetric algorithms that have an algorithm ID assigned by the TCG.

Table 61 — Definition of (TPM_ALG_ID) TPMI_ALG_ASYM Type

Values	Comments
TPM_ALG_!ALG.AO	all asymmetric object types
+TPM_ALG_NULL	
#TPM_RC_ASYMMETRIC	

9.25 TPMI_ALG_SYM (Symmetric Algorithms)

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

NOTE The validation code produced by an example script will produce a CASE statement with a case for each of the values in the "Values" column. The case for a value is delimited by a #ifdef/#endif pair so that if the algorithm is not implemented on the TPM, then the case for the algorithm is not generated, and use of the algorithm will cause a TPM error (TPM_RC_SYMMETRIC).

Table 62 — Definition of (TPM_ALG_ID) TPMI_ALG_SYM Type

Values	Comments
TPM_ALG_!ALG.S	all symmetric block ciphers
TPM_ALG_XOR	required
+TPM_ALG_NULL	required to be present in all versions of this table
#TPM_RC_SYMMETRIC	

9.26 TPMI_ALG_SYM_OBJECT

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

NOTE TPM_ALG_XOR is not allowed in this list.

Table 63 — Definition of (TPM_ALG_ID) TPMI_ALG_SYM_OBJECT Type

Values	Comments
TPM_ALG_!ALG.S	all symmetric block ciphers
+TPM_ALG_NULL	required to be present in all versions of this table
#TPM_RC_SYMMETRIC	

9.27 TPMI_ALG_SYM_MODE

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

Table 64 — Definition of (TPM_ALG_ID) TPMI_ALG_SYM_MODE Type

Values	Comments
TPM_ALG_!ALG.SE	all symmetric block cipher encryption/decryption modes
+TPM_ALG_NULL	
#TPM_RC_MODE	

9.28 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 65 — Definition of (TPM_ALG_ID) TPMI_ALG_KDF Type

Values	Comments
TPM_ALG_!ALG.HM	all defined hash-based key and mask generation functions
+TPM_ALG_NULL	
#TPM_RC_KDF	

9.29 TPMI_ALG_SIG_SCHEME

This is the definition of the interface type for any signature scheme.

Table 66 — Definition of (TPM_ALG_ID) TPMI_ALG_SIG_SCHEME Type

Values	Comments
TPM_ALG_!ALG.ax	all asymmetric signing schemes including anonymous schemes
TPM_ALG_HMAC	present on all TPM
+TPM_ALG_NULL	
#TPM_RC_SCHEME	response code when a signature scheme is not correct

9.30 TPMI_ECC_KEY_EXCHANGE

This is the definition of the interface type for an ECC key exchange scheme.

NOTE Because of the “{ECC}” in the table title, the only values in this table will be those that are dependent on ECC being implemented, even if they otherwise have the correct type attributes.

Table 67 — Definition of (TPM_ALG_ID){ECC} TPMI_ECC_KEY_EXCHANGE Type

Values	Comments
TPM_ALG_!ALG.AM	any ECC key exchange method
TPM_ALG_SM2	SM2 is typed as signing but may be used as a key-exchange protocol
+TPM_ALG_NULL	
#TPM_RC_SCHEME	response code when a key exchange scheme is not correct

9.31 TPMI_ST_COMMAND_TAG

This interface type is used for the command tags.

The response code for a bad command tag has the same value as the TPM 1.2 response code (TPM_BAD_TAG). This value is used in case the software is not compatible with this specification and an unexpected response code might have unexpected side effects.

Table 68 — Definition of (TPM_ST) TPMI_ST_COMMAND_TAG Type

Values	Comments
TPM_ST_NO_SESSIONS	
TPM_ST_SESSIONS	
#TPM_RC_BAD_TAG	

10 Structure Definitions

10.1 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.

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 69 — Definition of TPMS_EMPTY Structure <IN/OUT>

Parameter	Type	Description
		a structure with no member

10.2 TPMS_ALGORITHM_DESCRIPTION

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

Table 70 — Definition of TPMS_ALGORITHM_DESCRIPTION Structure <OUT>

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

10.3 Hash/Digest Structures

10.3.1 TPMU_HA (Hash)

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

NOTE 1 The !ALG.H and !ALG.H values represent all algorithms defined in the TCG registry as being type “H”.

NOTE 2 If processed by an automated tool, each entry of the table should be qualified (with #ifdef/#endif) so that if the hash algorithm is not implemented on the TPM, the parameter associated with that hash is not present. This will keep the union from being larger than the largest digest of a hash implemented on that TPM.

Table 71 — Definition of TPMU_HA Union <IN/OUT, S>

Parameter	Type	Selector	Description
!ALG.H [!ALG.H_DIGEST_SIZE]	BYTE	TPM_ALG_!ALG.H	all hashes
null		TPM_ALG_NULL	

10.3.2 TPMT_HA

Table 72 shows the basic hash-agile structure used in this specification. To handle hash agility, this structure uses the *hashAlg* parameter to indicate the algorithm used to compute the digest and, by implication, the size of the digest.

When transmitted, only the number of octets indicated by *hashAlg* is sent.

NOTE In the reference code, when a TPMT_HA is allocated, the digest field is large enough to support the largest hash algorithm in the TPMU_HA union.

Table 72 — Definition of TPMT_HA Structure <IN/OUT>

Parameter	Type	Description
hashAlg	+TPMI_ALG_HASH	selector of the hash contained in the *digest* that implies the size of the *digest* NOTE The leading "+" on the type indicates that this structure should pass an indication to the unmarshaling function for TPMI_ALG_HASH so that TPM_ALG_NULL will be allowed if a use of a TPMT_HA allows TPM_ALG_NULL.
[hashAlg] digest	TPMU_HA	the digest data

10.4 Sized Buffers

10.4.1 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 may be put into a TPM2B_ is that the parts of the structure may be handled as separate opaque blocks by the application/system software. Rather than require that all of the structures in a command or response be marshaled or unmarshaled sequentially, the size field allows the structure to be manipulated as an opaque block. Placing a structure in a TPM2B_ also makes it possible to use parameter encryption on the structure.

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

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

10.4.2 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 73 — 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

10.4.3 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.

Table 74 — 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	

10.4.4 TPM2B_NONCE

Table 75 — Definition of Types for TPM2B_NONCE

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

10.4.5 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 76 — Definition of Types for TPM2B_AUTH

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

10.4.6 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 77 — Definition of Types for TPM2B_OPERAND

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

10.4.7 TPM2B_EVENT

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

Table 78 — Definition of TPM2B_EVENT Structure

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

10.4.8 TPM2B_MAX_BUFFER

This type is a sized buffer that can hold a maximally sized buffer for commands that use a large data buffer such as TPM2_Hash(), TPM2_SequenceUpdate(), or TPM2_FieldUpgradeData().

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

Table 79 — 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.

10.4.9 TPM2B_MAX_NV_BUFFER

This type is a sized buffer that can hold a maximally sized buffer for NV data commands such as TPM2_NV_Read(), TPM2_NV_Write(), and TPM2_NV_Certify().

Table 80 — 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

10.4.10 TPM2B_TIMEOUT

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

Table 81 — Definition of Types for TPM2B_TIMEOUT

Type	Name	Description
TPM2B_DIGEST	TPM2B_TIMEOUT	size limited to the same as the digest structure

10.4.11 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 82 — Definition of TPM2B_IV Structure <IN/OUT>

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

10.5 Names

10.5.1 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.

10.5.2 TPMU_NAME

Table 83 — Definition of TPMU_NAME Union <>

Parameter	Type	Selector	Description
digest	TPMT_HA		when the Name is a digest
handle	TPM_HANDLE		when the Name is a handle

10.5.3 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 84 — Definition of TPM2B_NAME Structure

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

10.6 PCR Structures

10.6.1 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 value for *sizeofSelect* is determined by the number of PCR required by the applicable platform-specific specification and the number of PCR implemented in the TPM. The minimum value for *sizeofSelect* is:

$$\text{PCR\_SELECT\_MIN} := (\text{PLATFORM\_PCR} + 7) / 8 \quad (1)$$

where

PLATFORM_PCR 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 85 — 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		

10.6.2 TPMS_PCR_SELECTION

Table 86 — 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		

10.7 Tickets

10.7.1 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 10.7 are to be used by a TPM that is compliant with this specification.

The method of creating the ticket data is:

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

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 87.

Table 87 — Values for *proof* Used in Tickets

Hierarchy	proof	Description
Null	nullProof	a value that changes with every TPM Reset
Platform	phProof	a value that changes with each change of the PPS
Owner	shProof	a value that changes with each change of the SPS
Endorsement	ehProof	a value that changes with each change of either the EPS or SPS

The format for a ticket is shown in Table 88. This is a template for the tickets shown in the remainder of this clause 10.7.

Table 88 — 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

10.7.2 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 requires that the structure tag for the ticket be appropriate for the use, there is no single representation of a NULL Ticket that will work in all circumstances. Minimally, a NULL ticket will have a structure type that is appropriate for the context.

10.7.3 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 89 — 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>.

10.7.4 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 90 — 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>.

10.7.5 TPMT_TK_AUTH

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

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

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
cpHash	optional hash of the authorized command
policyRef	optional reference to a policy value
authName	Name of the object that signed the authorization
timeout	implementation-specific value indicating when the authorization expires
timeEpoch	implementation-specific representation of the *timeEpoch* at the time the ticket was created

NOTE 1 Not included if *timeout* is zero.

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

NOTE 2 Not included it *timeout* is zero or if *nonceTPM* was include in the authorization.

Table 91 — 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>

10.7.6 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 92 — 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*.

10.8 Property Structures

10.8.1 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 93 — 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

10.8.2 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 94 — Definition of TPMS_TAGGED_PROPERTY Structure <OUT>

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

10.8.3 TPMS_TAGGED_PCR_SELECT

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

Table 95 — Definition of TPMS_TAGGED_PCR_SELECT Structure <OUT>

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

10.8.4 TPMS_TAGGED_POLICY

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

Table 96 — Definition of TPMS_TAGGED_POLICY Structure <OUT>

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

10.9 Lists

10.9.1 TPML_CC

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

Table 97 — 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

10.9.2 TPML_CCA

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

The values in the list are returned in TPMA_CC->*commandIndex* order (see Table 37) with vendor-specific commands returned after other commands. Because of the other attributes, the commands may not be returned in strict numerical order.

Table 98 — 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

10.9.3 TPML_ALG

This list is returned by TPM2_IncrementalSelfTest().

Table 99 — 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

10.9.4 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 100 — 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

10.9.5 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 101 — 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

10.9.6 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 may not be in a list because the TPM may not recognize the algorithm as being a hash and it may not know the digest size.

Table 102 — 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

10.9.7 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 103 — 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

10.9.8 TPML_ALG_PROPERTY

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

NOTE MAX_CAP_ALGS = MAX_CAP_DATA ./ sizeof(TPMS_ALG_PROPERTY).

Table 104 — 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

10.9.9 TPML_TAGGED_TPM_PROPERTY

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

NOTE MAX_TPM_PROPERTIES = MAX_CAP_DATA ./ sizeof(TPMS_TAGGED_PROPERTY).

Table 105 — 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

10.9.10 TPML_TAGGED_PCR_PROPERTY

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

NOTE MAX_PCR_PROPERTIES = MAX_CAP_DATA ./ sizeof(TPMS_TAGGED_PCR_SELECT).

Table 106 — 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

10.9.11 TPML_ECC_CURVE

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

NOTE MAX_ECC_CURVES = MAX_CAP_DATA ./ sizeof(TPM_ECC_CURVE).

Table 107 — 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

10.9.12 TPML_TAGGED_POLICY

This list is used to report the authorization policy values for permanent handles. This is list may be generated by TPM2_GetCapabiltiy(). A permanent handle that cannot have a policy is not included in the list.

NOTE MAX_TAGGED_POLICIES = MAX_CAP_DATA ./ sizeof(TPMS_TAGGED_POLICY).

Table 108 — Definition of TPML_TAGGED_POLICY Structure <OUT>

Parameter	Type	Description
count	UINT32	number of tagged policies A value of zero is allowed.
policies[count]{:MAX_TAGGED_POLICIES}	TPMS_TAGGED_POLICY	array of tagged policies

10.10 Capabilities Structures

It is required that each parameter in this union be a list (TPML).

The number of returned elements in each list is determined by the size of each list element and the maximum size set by the vendor as the capability buffer (MAX_CAP_BUFFER in TPM_PT_MAX_CAP_BUFFER). The maximum number of bytes in a list is:

```
MAX_CAP_DATA = (MAX_CAP_BUFFER – sizeof(TPM_CAP) – sizeof(UINT32)      (8)
```

The maximum number of entries is then the number of complete list elements that will fit in MAX_CAP_DATA.

EXAMPLE For a 1024-octet MAX_CAP_BUFFER a response containing a TPML_HANDLE could have (1024 - 4 – 4) / 4 = 254 handles.

10.10.1 TPMU_CAPABILITIES

Table 109 — Definition of TPMU_CAPABILITIES Union <OUT>

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

10.10.2 TPMS_CAPABILITY_DATA

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

Table 110 — Definition of TPMS_CAPABILITY_DATA Structure <OUT>

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

10.11 Clock/Counter Structures

10.11.1 TPMS_CLOCK_INFO

This structure is used in each of the attestation commands.

Table 111 — Definition of TPMS_CLOCK_INFO Structure

Parameter	Type	Description
clock	UINT64	time value in milliseconds that advances while the TPM is powered NOTE The interpretation of the time-origin (*clock*=0) is out of the scope of this specification, although Coordinated Universal Time (UTC) is expected to be a common convention. This structure element is used to report on the TPM's Clock value. 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_ClockSet().
resetCount	UINT32	number of occurrences of TPM Reset since the last TPM2_Clear()
restartCount	UINT32	number of times that TPM2_Shutdown() or _TPM_Hash_Start have occurred since the last TPM Reset or TPM2_Clear().
safe	TPMI_YES_NO	no value of *Clock* greater than the current value of *Clock* has been previously reported by the TPM. Set to YES on TPM2_Clear().

10.11.2 *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.

10.11.3 *ResetCount*

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

10.11.4 *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().

10.11.5 *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().

10.11.6 TPMS_TIME_INFO

This structure is used in the TPM2_GetTime() 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 112 — 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

10.12 TPM Attestation Structures

10.12.1 Introduction

Clause 10.12 describes the structures that are used when a TPM creates a structure to be signed. The signing structures follow a standard format TPM2B_ATTEST with case-specific information embedded.

10.12.2 TPMS_TIME_ATTEST_INFO

This structure is used when the TPM performs TPM2_GetTime.

Table 113 — Definition of TPMS_TIME_ATTEST_INFO Structure <OUT>

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

10.12.3 TPMS_CERTIFY_INFO

This is the attested data for TPM2_Certify().

Table 114 — 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

10.12.1 TPMS_QUOTE_INFO

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

Table 115 — 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

10.12.2 TPMS_COMMAND_AUDIT_INFO

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

Table 116 — 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*

10.12.3 TPMS_SESSION_AUDIT_INFO

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

Table 117 — Definition of TPMS_SESSION_AUDIT_INFO Structure <OUT>

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

10.12.4 TPMS_CREATION_INFO

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

Table 118 — Definition of TPMS_CREATION_INFO Structure <OUT>

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

10.12.5 TPMS_NV_CERTIFY_INFO

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

Table 119 — 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

10.12.6 TPMI_ST_ATTEST

Table 120 — 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()

10.12.7 TPMU_ATTEST

Table 121 — 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

10.12.8 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 122 — Definition of TPMS_ATTEST Structure <OUT>

Parameter	Type	Description
magic	TPM_GENERATED	the indication that this structure was created by a TPM (always TPM_GENERATED_VALUE)
type	TPMI_ST_ATTEST	type of the attestation structure
qualifiedSigner	TPM2B_NAME	Qualified Name of the signing key
extraData	TPM2B_DATA	external information supplied by caller NOTE A TPM2B_DATA structure provides room for a digest and a method indicator to indicate the components of the digest. The definition of this method indicator is outside the scope of this specification.
clockInfo	TPMS_CLOCK_INFO	Clock, resetCount, restartCount, and Safe
firmwareVersion	UINT64	TPM-vendor-specific value identifying the version number of the firmware
[type]attested	TPMU_ATTEST	the type-specific attestation information

10.12.9 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 123 — 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

10.13 Authorization Structures

10.13.1 Introduction

The structures in 10.13 are used for all authorizations. One or more of these structures will be present in a command or response that has a tag of TPM_ST_SESSIONS.

10.13.2 TPMS_AUTH_COMMAND

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

Table 124 — 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

10.13.3 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 125 — Definition of TPMS_AUTH_RESPONSE Structure <OUT>

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

11 Algorithm Parameters and Structures

11.1 Symmetric

11.1.1 Introduction

Clause 11.1 defines the parameters and structures for describing symmetric algorithms.

11.1.2 TPMI_!ALG.S_KEY_BITS

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

Each implemented algorithm would have a value for the implemented key sizes for that implemented algorithm. That value would have a name in the form !ALG_KEY_SIZES_BITS where “ALG” would represent the characteristic name of the algorithm (such as “AES).

NOTE 1 Key size is expressed in bits.

Table 126 — Definition of {!ALG.S} (TPM_KEY_BITS) TPMI_!ALG.S_KEY_BITS Type

Parameter	Description
$!ALG.S_KEY_SIZES_BITS	number of bits in the key
#TPM_RC_VALUE	error when key size is not supported

11.1.3 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 127 — Definition of TPMU_SYM_KEY_BITS Union

Parameter	Type	Selector	Description
!ALG.S	TPMI_!ALG.S_KEY_BITS	TPM_ALG_!ALG.S	all symmetric algorithms
sym	TPM_KEY_BITS		when selector may be any of the symmetric block ciphers
xor	TPMI_ALG_HASH	TPM_ALG_XOR	overload for using *xor* NOTE TPM_ALG_NULL is not allowed
null		TPM_ALG_NULL	

11.1.4 TPMU_SYM_MODE

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

Table 128 — Definition of TPMU_SYM_MODE Union

Parameter	Type	Selector	Description
!ALG.S	TPMI_ALG_SYM_MODE+	TPM_ALG_!ALG.S	
sym	TPMI_ALG_SYM_MODE+		when selector may be any of the symmetric block ciphers
xor		TPM_ALG_XOR	no mode selector
null		TPM_ALG_NULL	no mode selector

11.1.5 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 129 —xDefinition of TPMU_SYM_DETAILS Union

Parameter	Type	Selector	Description
!ALG.S		TPM_ALG_!ALG	
sym			when selector may be any of the symmetric block ciphers
xor		TPM_ALG_XOR	
null		TPM_ALG_NULL	

11.1.6 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 130 — Definition of TPMT_SYM_DEF Structure

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

11.1.7 TPMT_SYM_DEF_OBJECT

This structure is used when different symmetric block cipher (not XOR) algorithms may be selected. If the Object can be an ordinary parent (not a derivation parent), this must be the first field in the Object's parameter (see 12.2.3.7) field.

Table 131 — Definition of TPMT_SYM_DEF_OBJECT Structure

Parameter	Type	Description
algorithm	+TPMI_ALG_SYM_OBJECT	selects a symmetric block cipher When used in the parameter area of a parent object, this shall be a supported block cipher and not TPM_ALG_NULL
[algorithm]keyBits	TPMU_SYM_KEY_BITS	the key size
[algorithm]mode	TPMU_SYM_MODE	default mode When used in the parameter area of a parent object, this shall be TPM_ALG_CFB.
//[algorithm]details	TPMU_SYM_DETAILS	contains the additional algorithm details, if any NOTE This is commented out at this time as the parser may not produce the proper code for a union if none of the selectors produces any data.

11.1.8 TPM2B_SYM_KEY

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

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

NOTE MAX_SYM_KEY_BYTES will be the larger of the largest symmetric key supported by the TPM and the largest digest produced by any hashing algorithm implemented on the TPM.

Table 132 — Definition of TPM2B_SYM_KEY Structure

Parameter	Type	Description
size	UINT16	size, in octets, of the buffer containing the key; may be zero
buffer [size] {:MAX_SYM_KEY_BYTES}	BYTE	the key

11.1.9 TPMS_SYMCIPHER_PARMS

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

Table 133 — Definition of TPMS_SYMCIPHER_PARMS Structure

Parameter	Type	Description
sym	TPMT_SYM_DEF_OBJECT	a symmetric block cipher

11.1.10 TPM2B_LABEL

This buffer holds a *label* or *context* value. For interoperability and backwards compatibility, LABEL_MAX_BUFFER is the minimum of the largest digest on the device and the largest ECC parameter (MAX_ECC_KEY_BYTES) but no more than 32 bytes.

All implementations are required to support at least one hash algorithm that produces a digest of 32 bytes or larger; and any implementation that supports ECC is required to support at least one curve with a key size of 32-bytes or larger.

NOTE Although the maximum size allowed for a *label* or *context* is 32 bytes, the object data structure needs to be sized to allow a 32-byte value.

Table 134 — Definition of TPM2B_LABEL Structure

Parameter	Type	Description
size	UINT16	
buffer[size]{:LABEL_MAX_BUFFER}	BYTE	symmetic data for a created object or the *label* and *context* for a derived object

11.1.11 TPMS_DERIVE

This structure contains the *label* and *context* fields for a derived object. These values are used in the derivation KDF. The values in the *unique* field of *inPublic* area template take precedence over the values in the *inSensitive* parameter.

Table 135 — Definition of TPMS_DERIVE Structure

Parameter	Type	Description
label	TPM2B_LABEL	
context	TPM2B_LABEL	

11.1.12 TPM2B_DERIVE

Table 136 — Definition of TPM2B_DERIVE Structure

Parameter	Type	Description
size	UINT16	
buffer[size]{: sizeof(TPMS_DERIVE)}	BYTE	symmetic data for a created object or the *label* and *context* for a derived object

11.1.13 TPMU_SENSITIVE_CREATE

This structure allows a TPM2B_SENSITIVE_CREATE structure to carry either a TPM2B_SENSITVE_DATA or a TPM2B_DERIVE structure. The contents of the union are determined by context. When an object is being derived, the derivation values are present.

MAX_SYM_DATA shall be 128.

NOTE No marshaling code is automatically generated for this union as it has no selectors that would allow the code to know the context and which member to unmarshal.

Table 137 — Definition of TPMU_SENSITIVE_CREATE Union <>

Parameter	Type	Selector	Description
create[MAX_SYM_DATA]	BYTE		sensitive data for a created symmetric Object
derive	TPMS_DERIVE		*label* and *context* for a derived Object

11.1.14 TPM2B_SENSITIVE_DATA

This buffer wraps the TPMU_SENSITIVE_CREATE structure.

Table 138 — Definition of TPM2B_SENSITIVE_DATA Structure

Parameter	Type	Description
size	UINT16	
buffer[size]{: sizeof(TPMU_SENSITIVE_CREATE)}	BYTE	symmetic data for a created object or the *label* and *context* for a derived object

11.1.15 TPMS_SENSITIVE_CREATE

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

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

If *data.size* is not zero, and the object is not a *keyedHash*, *data.size* must match the size indicated in the *keySize o*f *public.parameters.* If the object is a *keyedHash*, *data*.*size* may be any value up to the maximum allowed in a TPM2B_SENSITIVE_DATA.

For an asymmetric object, data shall be an Empty Buffer and *sensitiveDataOrigin* shall be SET.

Table 139 — Definition of TPMS_SENSITIVE_CREATE Structure <IN>

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

11.1.16 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 140 — Definition of TPM2B_SENSITIVE_CREATE Structure <IN, S>

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

11.1.17 TPMS_SCHEME_HASH

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

Table 141 — Definition of TPMS_SCHEME_HASH Structure

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

11.1.18 TPMS_SCHEME_ECDAA

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

Table 142 — 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

11.1.19 TPMI_ALG_HASH_SCHEME

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

Table 143 — Definition of (TPM_ALG_ID) TPMI_ALG_KEYEDHASH_SCHEME Type

Values	Comments
TPM_ALG_HMAC	the "signing" scheme
TPM_ALG_XOR	the "obfuscation" scheme
+TPM_ALG_NULL	
#TPM_RC_VALUE	

11.1.20 HMAC_SIG_SCHEME

Table 144 — Definition of Types for HMAC_SIG_SCHEME

Type	Name	Description
TPMS_SCHEME_HASH	TPMS_SCHEME_HMAC	

11.1.21 TPMS_SCHEME_XOR

This structure is for the XOR encryption scheme.

Table 145 — Definition of TPMS_SCHEME_XOR Structure

Parameter	Type	Description
hashAlg	TPMI_ALG_HASH+	the hash algorithm used to digest the message
kdf	TPMI_ALG_KDF+	the key derivation function

11.1.22 TPMU_SCHEME_KEYEDHASH

Table 146 — 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	

11.1.23 TPMT_KEYEDHASH_SCHEME

This structure is used for a hash signing object.

Table 147 — Definition of TPMT_KEYEDHASH_SCHEME Structure

Parameter	Type	Description
scheme	+TPMI_ALG_KEYEDHASH_SCHEME	selects the scheme
[scheme]details	TPMU_SCHEME_KEYEDHASH	the scheme parameters

11.2 Asymmetric

11.2.1 Signing Schemes

11.2.1.1 Introduction

These structures are used to define the method in which the signature is to be created. These schemes would appear in an object's public area and in commands where the signing scheme is variable.

Every scheme is required to indicate a hash that is used in digesting the message.

11.2.1.2 RSA Signature Schemes

These are the 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 148 — Definition of {RSA} Types for RSA Signature Schemes

Type	Name	Description
TPMS_SCHEME_HASH	TPMS_SIG_SCHEME_!ALG.AX	

11.2.1.3 ECC Signature Schemes

Most of the ECC signature schemes only require a hash algorithm to complete the definition and can be typed as TPMS_SCHEME_HASH. Anonymous algorithms also require a count value so they are typed to be TPMS_SCHEME_ECDAA.

Table 149 — Definition of {ECC} Types for ECC Signature Schemes

Type	Name	Description
TPMS_SCHEME_HASH	TPMS_SIG_SCHEME_!ALG.AX	all asymmetric signing schemes
TPMS_SCHEME_ECDAA	TPMS_SIG_SCHEME_!ALG.AXN	schemes that need a hash and a count

11.2.1.4 TPMU_SIG_SCHEME

The union of all of the signature schemes.

NOTE The TPMS_SIG_SCHEME_!ALG is determined by Table 148 or Table 149 and will be either a TPMS_SCHEME_HASH or a TPMS_SCHEME_ECDAA.

Table 150 — Definition of TPMU_SIG_SCHEME Union <IN/OUT, S>

Parameter	Type	Selector	Description
!ALG.ax	TPMS_SIG_SCHEME_!ALG	TPM_ALG_!ALG	all signing schemes including anonymous schemes
hmac	TPMS_SCHEME_HMAC	TPM_ALG_HMAC	the HMAC scheme
any	TPMS_SCHEME_HASH		selector that allows access to digest for any signing scheme
null		TPM_ALG_NULL	no scheme or default

11.2.1.5 TPMT_SIG_SCHEME

Table 151 — Definition of TPMT_SIG_SCHEME Structure

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

11.2.2 Encryption Schemes

11.2.2.1 Introduction

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

NOTE With ECC, the only encryption is with a key exchange of a symmetric key or seed.

11.2.2.2 RSA Encryption Schemes

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

NOTE: These types do not appear in the reference code in the specification but are used in the unmarshaling code.

Table 152 — Definition of Types for {RSA} Encryption Schemes

Type	Name	Description
TPMS_SCHEME_HASH	TPMS_ENC_SCHEME_!ALG.AEH	schemes that only need a hash
TPMS_EMPTY	TPMS_ENC_SCHEME_!ALG.AE	schemes that need nothing

11.2.2.3 ECC Key Exchange Schemes

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

NOTE: These types do not appear in the reference code in the specification but are used in the unmarshaling code.

Table 153 — Definition of Types for {ECC} ECC Key Exchange

Type	Name	Description
TPMS_SCHEME_HASH	TPMS_KEY_SCHEME_!ALG.AM	schemes that need a hash

11.2.3 Key Derivation Schemes

11.2.3.1 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.

The qualifying value for each of the kdf schemes is the hash algorithm.

NOTE: These types do not appear in the reference code in the specification but are used in the unmarshaling code.

Table 154 — Definition of Types for KDF Schemes

Type	Name	Description
TPMS_SCHEME_HASH	TPMS_SCHEME_!ALG.HM	hash-based key- or mask-generation functions

11.2.3.2 TPMU_KDF_SCHEME

Table 155 — Definition of TPMU_KDF_SCHEME Union <IN/OUT, S>

Parameter	Type	Selector	Description
!ALG.HM	TPMS_SCHEME_!ALG.HM	TPM_ALG_!ALG.HM	
null		TPM_ALG_NULL	

11.2.3.3 TPMT_KDF_SCHEME

Table 156 — Definition of TPMT_KDF_SCHEME Structure

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

11.2.3.4 TPMI_ALG_ASYM_SCHEME

List of all of the scheme types for any asymmetric algorithm.

NOTE 1 This is the selector value used to define TPMT_ASYM_SCHEME.

NOTE 2 Most tokens are exclusive in order to filter out SM2 and other multi-protocol algorithm identifiers. The inclusive token “ax” will include those algorithms.

Table 157 — Definition of (TPM_ALG_ID) TPMI_ALG_ASYM_SCHEME Type <>

Values	Comments
TPM_ALG_!ALG.am	key exchange methods
TPM_ALG_!ALG.ax	all signing including anonymous
TPM_ALG_!ALG.ae	encrypting schemes
+TPM_ALG_NULL	
#TPM_RC_VALUE	

11.2.3.5 TPMU_ASYM_SCHEME

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

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

Table 158 — Definition of TPMU_ASYM_SCHEME Union

Parameter	Type	Selector	Description
!ALG.am	TPMS_KEY_SCHEME_!ALG	TPM_ALG_!ALG	
!ALG.ax	TPMS_SIG_SCHEME_!ALG	TPM_ALG_!ALG	signing and anonymous signing
!ALG.ae	TPMS_ENC_SCHEME_!ALG	TPM_ALG_!ALG	schemes with no hash
anySig	TPMS_SCHEME_HASH		
null		TPM_ALG_NULL	no scheme or default This selects the NULL Signature.

11.2.3.6 TPMT_ASYM_SCHEME

This structure is defined to allow overlay of all of the schemes for any asymmetric object. This structure is not sent on the interface. It is defined so that common functions may operate on any similar scheme structure.

EXAMPLE Since many schemes have a hash algorithm as their defining parameter, a common function can use the digest selector to select the hash of the scheme without a need to cast or use a large switch statement.

Table 159 — Definition of TPMT_ASYM_SCHEME Structure <>

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

11.2.4 RSA

11.2.4.1 TPMI_ALG_RSA_SCHEME

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

Table 160 — Definition of (TPM_ALG_ID) {RSA} TPMI_ALG_RSA_SCHEME Type

Values	Comments
TPM_ALG_!ALG.ae.ax	encrypting and signing algorithms
+TPM_ALG_NULL	
#TPM_RC_VALUE	

11.2.4.2 TPMT_RSA_SCHEME

Table 161 — Definition of {RSA} TPMT_RSA_SCHEME Structure

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

11.2.4.3 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 162 — Definition of (TPM_ALG_ID) {RSA} TPMI_ALG_RSA_DECRYPT Type

Values	Comments
TPM_ALG_!ALG.ae	all RSA encryption algorithms
+TPM_ALG_NULL	
#TPM_RC_VALUE	

11.2.4.4 TPMT_RSA_DECRYPT

Table 163 — Definition of {RSA} TPMT_RSA_DECRYPT Structure

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

11.2.4.5 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 164 — Definition of {RSA} TPM2B_PUBLIC_KEY_RSA Structure

Parameter	Type	Description
size	UINT16	size of the buffer The value of zero is only valid for create.
buffer[size] {: MAX_RSA_KEY_BYTES}	BYTE	Value

11.2.4.6 TPMI_RSA_KEY_BITS

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

NOTE 1 An implementation is allowed to provide limited support for smaller RSA key sizes. That is, a TPM may be able to accept a smaller RSA key size in TPM2_LoadExternal() when only the public area is loaded but not accept that smaller key size in any command that loads both the public and private portions of an RSA key. This would allow the TPM to validate signatures using the smaller key but would prevent the TPM from using the smaller key size for any other purpose.

NOTE 2 The definition for RSA_KEY_SIZES_BITS used in the reference implementation is found in TPM 2.0 Part 4, Implementation.h

Table 165 — Definition of {RSA} (TPM_KEY_BITS) TPMI_RSA_KEY_BITS Type

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

11.2.4.7 TPM2B_PRIVATE_KEY_RSA

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

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

Table 166 — Definition of {RSA} TPM2B_PRIVATE_KEY_RSA Structure

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

11.2.5 ECC

11.2.5.1 TPM2B_ECC_PARAMETER

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

Table 167 — Definition of TPM2B_ECC_PARAMETER Structure

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

11.2.5.2 TPMS_ECC_POINT

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

Table 168 — Definition of {ECC} TPMS_ECC_POINT Structure

Parameter	Type	Description
x	TPM2B_ECC_PARAMETER	X coordinate
y	TPM2B_ECC_PARAMETER	Y coordinate

11.2.5.3 TPM2B_ECC_POINT

This 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 169 — 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*

11.2.5.4 TPMI_ALG_ECC_SCHEME

Table 170 — Definition of (TPM_ALG_ID) {ECC} TPMI_ALG_ECC_SCHEME Type

Values	Comments
TPM_ALG_!ALG.ax	the ecc signing schemes
TPM_ALG_!ALG.am	key exchange methods
+TPM_ALG_NULL	
#TPM_RC_SCHEME	

11.2.5.5 TPMI_ECC_CURVE

This type enumerates the ECC curves implemented by the TPM.

Table 171 — 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

11.2.5.6 TPMT_ECC_SCHEME

Table 172 — 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

11.2.5.7 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 173 — Definition of {ECC} TPMS_ALGORITHM_DETAIL_ECC Structure <OUT>

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

11.3 Signatures

11.3.1 TPMS_SIGNATURE_RSA

Table 174 — Definition of {RSA} TPMS_SIGNATURE_RSA Structure

Parameter	Type	Description
hash	TPMI_ALG_HASH	the hash algorithm used to digest the message TPM_ALG_NULL is not allowed.
sig	TPM2B_PUBLIC_KEY_RSA	The signature is the size of a public key.

Table 175 — Definition of Types for {RSA} Signature

Type	Name	Description
TPMS_SIGNATURE_RSA	TPMS_SIGNATURE_!ALG.ax	

11.3.2 TPMS_SIGNATURE_ECC

Table 176 — Definition of {ECC} TPMS_SIGNATURE_ECC Structure

Parameter	Type	Description
hash	TPMI_ALG_HASH	the hash algorithm used in the signature process TPM_ALG_NULL is not allowed.
signatureR	TPM2B_ECC_PARAMETER	
signatureS	TPM2B_ECC_PARAMETER	

Table 177 — Definition of Types for {ECC} TPMS_SIGNATURE_ECC

Type	Name	Description
TPMS_SIGNATURE_ECC	TPMS_SIGNATURE_!ALG.ax	

11.3.3 TPMU_SIGNATURE

A TPMU_SIGNATURE_COMPOSITE is a union of the various signatures that are supported by a particular TPM implementation. The union allows substitution of any signature algorithm wherever a signature is required in a structure.

NOTE All TPM are required to support a hash algorithm and the HMAC algorithm.

When a symmetric algorithm is used for signing, the signing algorithm is assumed to be an HMAC based on the indicated hash algorithm. The HMAC key will either be referenced as part of the usage or will be implied by context.

Table 178 — Definition of TPMU_SIGNATURE Union <IN/OUT, S>

Parameter	Type	Selector	Description
!ALG.ax	TPMS_SIGNATURE_!ALG.ax	TPM_ALG_!ALG.ax	all asymmetric signatures
hmac	TPMT_HA	TPM_ALG_HMAC	HMAC signature (required to be supported)
any	TPMS_SCHEME_HASH		used to access the hash
null		TPM_ALG_NULL	the NULL signature

11.3.4 TPMT_SIGNATURE

Table 179 shows the basic algorithm-agile structure when a symmetric or asymmetric signature is indicated. The *sigAlg* parameter indicates the algorithm used for the signature. This structure is output from the attestation commands and is an input to TPM2_VerifySignature(), TPM2_PolicySigned(), and TPM2_FieldUpgradeStart().

Table 179 — Definition of TPMT_SIGNATURE Structure

Parameter	Type	Description
sigAlg	+TPMI_ALG_SIG_SCHEME	selector of the algorithm used to construct the signature
[sigAlg]signature	TPMU_SIGNATURE	This shall be the actual signature information.

11.4 Key/Secret Exchange

11.4.1 Introduction

The structures in 11.4 are used when a key or secret is being exchanged. The exchange may be in

- TPM2_StartAuthSession() where the secret is injected for salting the session,
- TPM2_Duplicate(), TPM2_Import, or TPM2_Rewrap() where the secret is the symmetric encryption key for the outer wrapper of a duplication blob, or
- TPM2_ActivateIdentity() or TPM2_CreateIdentity() where the secret is the symmetric encryption key for the credential blob.

Particulars are described in TPM 2.0 Part 1.

11.4.2 TPMU_ENCRYPTED_SECRET

This structure is used to hold either an ephemeral public point for ECDH, an OAEP-encrypted block for RSA, or a symmetrically encrypted value. This structure is defined for the limited purpose of determining the size of a TPM2B_ENCRYPTED_SECRET.

The symmetrically encrypted value may use either CFB or XOR encryption.

NOTE Table 180 is illustrative. It would be modified depending on the algorithms supported in the TPM.

Table 180 — Definition of TPMU_ENCRYPTED_SECRET Union <S>

Parameter	Type	Selector	Description
ecc[sizeof(TPMS_ECC_POINT)]	BYTE	TPM_ALG_ECC	
rsa[MAX_RSA_KEY_BYTES]	BYTE	TPM_ALG_RSA	
symmetric[sizeof(TPM2B_DIGEST)]	BYTE	TPM_ALG_SYMCIPHER	
keyedHash[sizeof(TPM2B_DIGEST)]	BYTE	TPM_ALG_KEYEDHASH	Any symmetrically encrypted secret value will be limited to be no larger than a digest.

11.4.3 TPM2B_ENCRYPTED_SECRET

Table 181 — Definition of TPM2B_ENCRYPTED_SECRET Structure

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

12 Key/Object Complex

12.1 Introduction

An object description requires a TPM2B_PUBLIC structure and may require a TPMT_SENSITIVE structure. When the structure is stored off the TPM, the TPMT_SENSITIVE structure is encrypted within a TPM2B_PRIVATE structure.

When the object requires two components for its description, those components are loaded as separate parameters in the TPM2_Load() command. When the TPM creates an object that requires both components, the TPM will return them as separate parameters from the TPM2_Create() operation.

The TPM may produce multiple different TPM2B_PRIVATE structures for a single TPM2B_PUBLIC structure. Creation of a modified TPM2B_PRIVATE structure requires that the full structure be loaded with the TPM2_Load() command, modification of the TPMT_SENSITIVE data, and output of a new TPM2B_PRIVATE structure.

12.2 Public Area Structures

12.2.1 Description

Clause 12.2 defines the TPM2B_PUBLIC structure and the higher-level substructure that may be contained in a TPM2B_PUBLIC. The higher-level structures that are currently defined for inclusion in a TPM2B_PUBLIC are the

- structures for asymmetric keys,
- structures for symmetric keys, and
- structures for sealed data.

12.2.2 TPMI_ALG_PUBLIC

Table 182 — Definition of (TPM_ALG_ID) TPMI_ALG_PUBLIC Type

Values	Comments
TPM_ALG_!ALG.o	All object types
#TPM_RC_TYPE	response code when a public type is not supported

12.2.3 Type-Specific Parameters

12.2.3.1 Description

The public area contains two fields (*parameters* and *unique*) that vary by object type. The *parameters* field varies according to the *type* of the object but the contents may be the same across multiple instances of a particular *type*. The unique field format also varies according to the type of the object and will also be unique for each instance.

For a symmetric key (*type* == TPM_ALG_SYMCIPHER), HMAC key (*type* == TPM_ALG_KEYEDHASH) or data object (also, *type* == TPM_ALG_KEYEDHASH), the contents of *unique* shall be computed from components of the sensitive area of the object as follows:

$$unique := \mathbf{H}_{nameAlg}(seedValue \,||\, sensitive) \quad (9)$$

where

$\mathbf{H}_{nameAlg}()$	the hash algorithm used to compute the Name of the object
$seedValue$	the digest-sized obfuscation value in the sensitive area of a symmetric key or symmetric data object found in a TPMT_SENSITIVE.*seedValue*.*buffer*
$sensitive$	the secret key/data of the object in the TPMT_SENSITIVE.*sensitive*.*any*.*buffer*

12.2.3.2 TPMU_PUBLIC_ID

This is the union of all values allowed in in the ***unique*** field of a TPMT_PUBLIC.

NOTE The derive member cannot be unmarshaled in a TPMU_PUBLIC_ID. It is placed in this structure so that the maximum size of a TPM2B_TEMPLATE will be computed correctly.

Table 183 — Definition of TPMU_PUBLIC_ID Union <IN/OUT, S>

Parameter	Type	Selector	Description
keyedHash	TPM2B_DIGEST	TPM_ALG_KEYEDHASH	
sym	TPM2B_DIGEST	TPM_ALG_SYMCIPHER	
rsa	TPM2B_PUBLIC_KEY_RSA	TPM_ALG_RSA	
ecc	TPMS_ECC_POINT	TPM_ALG_ECC	
derive	TPMS_DERIVE		only allowed for TPM2_CreateLoaded when *parentHandle* is a Derivation Parent.

12.2.3.3 TPMS_KEYEDHASH_PARMS

This structure describes the parameters that would appear in the public area of a KEYEDHASH object.

NOTE Although the names are the same, the types of the structures are not the same as for asymmetric parameter lists.

Table 184 — Definition of TPMS_KEYEDHASH_PARMS Structure

Parameter	Type	Description
scheme	TPMT_KEYEDHASH_SCHEME+	Indicates the signing method used for a keyedHash signing object. This field also determines the size of the data field for a data object created with TPM2_Create() or TPM2_CreatePrimary().

12.2.3.4 TPMS_ASYM_PARMS

This structure contains the common public area parameters for an asymmetric key. The first two parameters of the parameter definition structures of an asymmetric key shall have the same two first components.

NOTE The sign parameter may have a different type in order to allow different schemes to be selected for each asymmetric type but the first parameter of each scheme definition shall be a TPM_ALG_ID for a valid signing scheme.

Table 185 — Definition of TPMS_ASYM_PARMS Structure <>

Parameter	Type	Description
symmetric	TPMT_SYM_DEF_OBJECT+	the companion symmetric algorithm for a restricted decryption key and shall be set to a supported symmetric algorithm This field is optional for keys that are not decryption keys and shall be set to TPM_ALG_NULL if not used.
scheme	TPMT_ASYM_SCHEME+	for a key with the *sign* attribute SET, a valid signing scheme for the key type for a key with the *decrypt* attribute SET, a valid key exchange protocol for a key with sign and decrypt attributes, shall be TPM_ALG_NULL

12.2.3.5 TPMS_RSA_PARMS

A TPM compatible with this specification and supporting RSA shall support two primes and an *exponent* of zero. Support for other values is optional. Use of other exponents in duplicated keys is not recommended because the resulting keys would not be interoperable with other TPMs.

NOTE Implementations are not required to check that *exponent* is the default exponent. They may fail to load the key if *exponent* is not zero. The reference implementation allows the values listed in the table.

Table 186 — Definition of {RSA} TPMS_RSA_PARMS Structure

Parameter	Type	Description
symmetric	TPMT_SYM_DEF_OBJECT+	for a restricted decryption key, shall be set to a supported symmetric algorithm, key size, and mode. if the key is not a restricted decryption key, this field shall be set to TPM_ALG_NULL.
scheme	TPMT_RSA_SCHEME+	scheme.scheme shall be: for an unrestricted signing key, either TPM_ALG_RSAPSS TPM_ALG_RSASSA or TPM_ALG_NULL for a restricted signing key, either TPM_ALG_RSAPSS or TPM_ALG_RSASSA for an unrestricted decryption key, TPM_ALG_RSAES, TPM_ALG_OAEP, or TPM_ALG_NULL unless the object also has the *sign* attribute for a restricted decryption key, TPM_ALG_NULL NOTE When both sign and decrypt are SET, restricted shall be CLEAR and scheme shall be TPM_ALG_NULL.
keyBits	TPMI_RSA_KEY_BITS	number of bits in the public modulus
exponent	UINT32	the public exponent A prime number greater than 2. When zero, indicates that the exponent is the default of $2^{16} + 1$

12.2.3.6 TPMS_ECC_PARMS

This structure contains the parameters for prime modulus ECC.

Table 187 — Definition of {ECC} TPMS_ECC_PARMS Structure

Parameter	Type	Description
symmetric	TPMT_SYM_DEF_OBJECT+	for a restricted decryption key, shall be set to a supported symmetric algorithm, key size. and mode. if the key is not a restricted decryption key, this field shall be set to TPM_ALG_NULL.
scheme	TPMT_ECC_SCHEME+	If the *sign* attribute of the key is SET, then this shall be a valid signing scheme. NOTE If the *sign* parameter in *curveID* indicates a mandatory scheme, then this field shall have the same value. If the *decrypt* attribute of the key is SET, then this shall be a valid key exchange scheme or TPM_ALG_NULL. If the key is a Storage Key, then this field shall be TPM_ALG_NULL.
curveID	TPMI_ECC_CURVE	ECC curve ID
kdf	TPMT_KDF_SCHEME+	an optional key derivation scheme for generating a symmetric key from a Z value If the *kdf* parameter associated with *curveID* is not TPM_ALG_NULL then this is required to be NULL. NOTE There are currently no commands where this parameter has effect and, in the reference code, this field needs to be set to TPM_ALG_NULL.

12.2.3.7 TPMU_PUBLIC_PARMS

Table 188 defines the possible parameter definition structures that may be contained in the public portion of a key. If the Object can be a parent, the first field must be a TPMT_SYM_DEF_OBJECT. See 11.1.7.

Table 188 — Definition of TPMU_PUBLIC_PARMS Union <IN/OUT, S>

Parameter	Type	Selector	Description(1)
keyedHashDetail	TPMS_KEYEDHASH_PARMS	TPM_ALG_KEYEDHASH	sign | decrypt | neither
symDetail	TPMS_SYMCIPHER_PARMS	TPM_ALG_SYMCIPHER	a symmetric block cipher
rsaDetail	TPMS_RSA_PARMS	TPM_ALG_RSA	decrypt + sign(2)
eccDetail	TPMS_ECC_PARMS	TPM_ALG_ECC	decrypt + sign(2)
asymDetail	TPMS_ASYM_PARMS		common scheme structure for RSA and ECC keys

NOTES
1) Description column indicates which of TPMA_OBJECT.*decrypt* or TPMA_OBJECT.*sign* may be set.
2) "+" indicates that both may be set but one shall be set. "|" indicates the optional settings.

12.2.3.8 TPMT_PUBLIC_PARMS

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

Table 189 — Definition of TPMT_PUBLIC_PARMS Structure

Parameter	Type	Description
type	TPMI_ALG_PUBLIC	the algorithm to be tested
[type]parameters	TPMU_PUBLIC_PARMS	the algorithm details

12.2.4 TPMT_PUBLIC

Table 190 defines the public area structure. The Name of the object is *nameAlg* concatenated with the digest of this structure using *nameAlg*.

Table 190 — Definition of TPMT_PUBLIC Structure

Parameter	Type	Description
type	TPMI_ALG_PUBLIC	“algorithm” associated with this object
nameAlg	+TPMI_ALG_HASH	algorithm used for computing the Name of the object NOTE The "+" indicates that the instance of a TPMT_PUBLIC may have a "+" to indicate that the *nameAlg* may be TPM_ALG_NULL.
objectAttributes	TPMA_OBJECT	attributes that, along with *type*, determine the manipulations of this object
authPolicy	TPM2B_DIGEST	optional policy for using this key The policy is computed using the *nameAlg* of the object. NOTE Shall be the Empty Policy if no authorization policy is present.
[type]parameters	TPMU_PUBLIC_PARMS	the algorithm or structure details
[type]unique	TPMU_PUBLIC_ID	the unique identifier of the structure For an asymmetric key, this would be the public key.

12.2.5 TPM2B_PUBLIC

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

Table 191 — Definition of TPM2B_PUBLIC Structure

Parameter	Type	Description
size=	UINT16	size of publicArea NOTE The “=” will force the TPM to try to unmarshal a TPMT_PUBLIC and check that the unmarshaled size matches the value of *size.* If all the required fields of a TPMT_PUBLIC are not present, the TPM will return an error (generally TPM_RC_SIZE) when attempting to unmarshal the TPMT_PUBLIC.
publicArea	+TPMT_PUBLIC	the public area NOTE The “+” indicates that the caller may specify that use of TPM_ALG_NULL is allowed for *nameAlg.*

12.2.6 TPM2B_TEMPLATE

This sized buffer is used to embed a TPMT_TEMPLATE for TPM2_CreateLoaded().

Unmarshaling of this structure is fairly complex due to requirements for backwards compatibility. Unlike a TPM2B_PUBLIC, this structure is unmarshaled as an array of bytes that is passed to the action code. The action code will then unmarshal the embedded structure.

For key derivation using TPM2_CreateLoaded(), the unmarshaling may return TPM_RC_SIZE because the size of the ***unique*** parameter is really the first of two size fields. This will cause the marshaling code to fail because the unmarshaling completes before all of the data in the buffer is read. If this occurs, the action code for TPM2_CreateLoaded() will verify that the remaining size is consistent with there being an additional TPM2B in the input.

Table 192 — Definition of TPM2B_TEMPLATE Structure

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

12.3 Private Area Structures

12.3.1 Introduction

The structures in 12.2.6 define the contents and construction of the private portion of a TPM object. A TPM2B_PRIVATE along with a TPM2B_PUBLIC are needed to describe a TPM object.

A TPM2B_PRIVATE area may be encrypted by different symmetric algorithms or, in some cases, not encrypted at all.

12.3.2 Sensitive Data Structures

12.3.2.1 Introduction

The structures in 12.3.2 define the presumptive internal representations of the sensitive areas of the various entities. A TPM may store the sensitive information in any desired format but when constructing a TPM_PRIVATE, the formats in 12.3.2 shall be used.

12.3.2.2 TPM2B_PRIVATE_VENDOR_SPECIFIC

This structure is defined for coding purposes. For IO to the TPM, the sensitive portion of the key will be in a canonical form. For an RSA key, this will be one of the prime factors of the public modulus. After loading, it is typical that other values will be computed so that computations using the private key will not need to start with just one prime factor. This structure can be used to store the results of such vendor-specific calculations.

The value for RSA_VENDOR_SPECIFIC is determined by the vendor.

Table 193 — Definition of TPM2B_PRIVATE_VENDOR_SPECIFIC Structure<>

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

12.3.2.3 TPMU_SENSITIVE_COMPOSITE

Table 194 — Definition of TPMU_SENSITIVE_COMPOSITE Union <IN/OUT, S>

Parameter	Type	Selector	Description
rsa	TPM2B_PRIVATE_KEY_RSA	TPM_ALG_RSA	a prime factor of the public key
ecc	TPM2B_ECC_PARAMETER	TPM_ALG_ECC	the integer private key
bits	TPM2B_SENSITIVE_DATA	TPM_ALG_KEYEDHASH	the private data
sym	TPM2B_SYM_KEY	TPM_ALG_SYMCIPHER	the symmetric key
any	TPM2B_PRIVATE_VENDOR_SPECIFIC		vendor-specific size for key storage

12.3.2.4 TPMT_SENSITIVE

Table 195 — Definition of TPMT_SENSITIVE Structure

Parameter	Type	Description
sensitiveType	TPMI_ALG_PUBLIC	identifier for the sensitive area This shall be the same as the *type* parameter of the associated public area.
authValue	TPM2B_AUTH	user authorization data The authValue may be a zero-length string. This value shall not be larger than the size of the digest produced by the *nameAlg* of the object.
seedValue	TPM2B_DIGEST	for a parent object, the optional protection seed; for other objects, the obfuscation value This value shall not be larger than the size of the digest produced by *nameAlg* of the object.
[sensitiveType]sensitive	TPMU_SENSITIVE_COMPOSITE	the type-specific private data

12.3.3 TPM2B_SENSITIVE

The TPM2B_SENSITIVE structure is used as a parameter in TPM2_LoadExternal(). It is an unencrypted sensitive area but it may be encrypted using parameter encryption.

NOTE 1 When this structure is unmarshaled, the *sensitiveType* determines what type of value is unmarshaled. Each value of *sensitiveType* is associated with a TPM2B. It is the maximum size for each of the TPM2B values that will determine if the unmarshal operation is successful. Since there is no selector for the *any* or *vendor* options for the union, the maximum input and output sizes for a TPM2B_SENSITIVE are not affected by the sizes of those parameters.

NOTE 2 The unmarshaling function validates that *size* equals the size of the value that is unmarshaled.

Table 196 — Definition of TPM2B_SENSITIVE Structure <IN/OUT>

Parameter	Type	Description
size	UINT16	size of the *private* structure
sensitiveArea	TPMT_SENSITIVE	an unencrypted sensitive area

12.3.4 Encryption

A TPMS_SENSITIVE is the input to the encryption process. All TPMS_ENCRYPT structures are CFB-encrypted using a key and Initialization Vector (IV) that are derived from a seed value.

The method of generating the key and IV is described in “Protected Storage” subclause “Symmetric Encryption.” in TPM 2.0 Part 1.

12.3.5 Integrity

The integrity computation is used to ensure that a protected object is not modified when stored in memory outside of the TPM.

The method of protecting the integrity of the sensitive area is described in “Protected Storage” subclause “Integrity” in TPM 2.0 Part 1.

12.3.6 _PRIVATE

This structure is defined to size the contents of a TPM2B_PRIVATE. This structure is not directly marshaled or unmarshaled.

For TPM2_Duplicate() and TPM2_Import(), the TPM2B_PRIVATE may contain multiply encrypted data and two integrity values. In some cases, the sensitive data is not encrypted and the integrity value is not present.

For TPM2_Load() and TPM2_Create(), *integrityInner* is always present.

If *integrityInner* is present, it and *sensitive* are encrypted as a single block.

When an integrity value is not needed, it is not present and it is not represented by an Empty Buffer.

Table 197 — Definition of _PRIVATE Structure <>

Parameter	Type	Description
integrityOuter	TPM2B_DIGEST	
integrityInner	TPM2B_DIGEST	could also be a TPM2B_IV
sensitive	TPM2B_SENSITIVE	the sensitive area

12.3.7 TPM2B_PRIVATE

The TPM2B_PRIVATE structure is used as a parameter in multiple commands that create, load, and modify the sensitive area of an object.

When the TPM returns a TPM2B_PRIVATE structure, the TPM pads the TPM2B_AUTH to its maximum size.

Table 198 — Definition of TPM2B_PRIVATE Structure <IN/OUT, S>

Parameter	Type	Description
size	UINT16	size of the *private* structure
buffer[size] {:sizeof(_PRIVATE)}	BYTE	an encrypted private area

12.4 Identity Object

12.4.1 Description

An identity object is used to convey credential protection value (CV) to a TPM that can load the object associated with the object. The CV is encrypted to a storage key on the target TPM, and if the credential integrity checks and the proper object is loaded in the TPM, then the TPM will return the CV.

12.4.2 TPMS_ID_OBJECT

This structure is used for sizing the TPM2B_ID_OBJECT.

Table 199 — Definition of TPMS_ID_OBJECT Structure <>

Parameter	Type	Description
integrityHMAC	TPM2B_DIGEST	HMAC using the nameAlg of the storage key on the target TPM
encIdentity	TPM2B_DIGEST	credential protector information returned if name matches the referenced object All of the *encIdentity* is encrypted, including the size field. NOTE The TPM is not required to check that the size is not larger than the digest of the *nameAlg*. However, if the size is larger, the ID object may not be usable on a TPM that has no digest larger than produced by *nameAlg*.

12.4.3 TPM2B_ID_OBJECT

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

Table 200 — Definition of TPM2B_ID_OBJECT Structure <IN/OUT>

Parameter	Type	Description
size	UINT16	size of the *credential* structure
credential[size]{:sizeof(TPMS_ID_OBJECT)}	BYTE	an encrypted credential area

13 NV Storage Structures

13.1 TPM_NV_INDEX

A TPM_NV_INDEX is used to reference a defined location in NV memory. The format of the Index is changed from TPM 1.2 in order to include the Index in the reserved handle space. Handles in this range use the digest of the public area of the Index as the Name of the entity in authorization computations

The 32-bit TPM 1.2 NV Index format is shown in Figure 4. In order to allow the Index to fit into the 24 bits available in the reserved handle space, the Index value format is changed as shown in Figure 5.

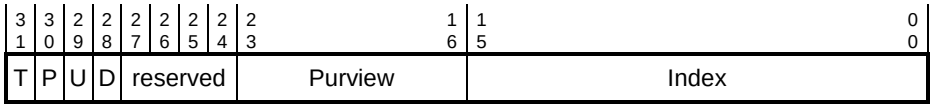

Figure 4 — TPM 1.2 TPM_NV_INDEX

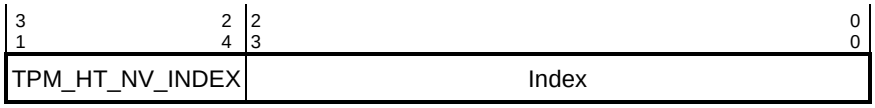

Figure 5 — TPM 2.0 TPM_NV_INDEX

NOTE This TPM_NV_INDEX format does not retain the Purview field and the D bit is not a part of an Index handle as in TPM 1.2. The TPMA_NV_PLATFORMCREATE attribute is a property of an Index that provides functionality similar to the D bit.

A valid Index handle will have an MSO of TPM_HT_NV_INDEX.

NOTE This structure is not used. It is defined here to indicate how the fields of the handle are assigned. The exemplary unmarshaling code unmarshals a TPM_HANDLE and validates that it is in the range for a TPM_NV_INDEX.

Table 201 — Definition of (UINT32) TPM_NV_INDEX Bits <>

Bit	Name	Definition
23:00	index	The Index of the NV location
31:24	RH_NV	constant value of TPM_HT_NV_INDEX indicating the NV Index range

Some prior versions of this specification contained a table here (Options for space Field of TPM_NV_INDEX) that assigned subsets of the index field to different entities. Since this assignment was a convention and not an architectural element of the TPM, the table was removed and the information is now contained in a registry document that is maintained by the TCG.

13.2 TPM_NT

This table lists the values of the TPM_NT field of a TPMA_NV. See Table 204 for usage.

Table 202 — Definition of TPM_NT Constants

Name	Value	Description
TPM_NT_ORDINARY	0x0	Ordinary – contains data that is opaque to the TPM that can only be modified using TPM2_NV_Write().
TPM_NT_COUNTER	0x1	Counter – contains an 8-octet value that is to be used as a counter and can only be modified with TPM2_NV_Increment()
TPM_NT_BITS	0x2	Bit Field – contains an 8-octet value to be used as a bit field and can only be modified with TPM2_NV_SetBits().
TPM_NT_EXTEND	0x4	Extend – contains a digest-sized value used like a PCR. The Index can only be modified using TPM2_NV_Extend(). The extend will use the nameAlg of the Index.
TPM_NT_PIN_FAIL	0x8	PIN Fail - contains *pinCount* that increments on a PIN authorization failure and a *pinLimit*
TPM_NT_PIN_PASS	0x9	PIN Pass - contains *pinCount* that increments on a PIN authorization success and a *pinLimit*

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

NOTE 1 These values are compatible with previous versions of this specification, which used a bit map for this field.

NOTE 2 This field is described by Table 202 is 4 bits.

13.3 TPMS_NV_PIN_COUNTER_PARAMETERS

This is the data that can be written to and read from a TPM_NT_PIN_PASS or TPM_NT_PIN_FAIL non-volatile index. *pinCount* is the most significant octets. *pinLimit* is the least significant octets.

Table 203 — Definition of TPMS_NV_PIN_COUNTER_PARAMETERS Structure

Parameter	Type	Description
pinCount	UINT32	This counter shows the current number of successful authValue authorization attempts to access a TPM_NT_PIN_PASS index or the current number of unsuccessful authValue authorization attempts to access a TPM_NT_PIN_FAIL index.
pinLimit	UINT32	This threshold is the value of *pinCount* at which the authValue authorization of the host TPM_NT_PIN_PASS or TPM_NT_PIN_FAIL index is locked out.

13.4 TPMA_NV (NV Index Attributes)

This structure allows the TPM to keep track of the data and permissions to manipulate an NV Index.

The platform controls (TPMA_NV_PPWRITE and TPMA_NV_PPREAD) and owner controls (TPMA_NV_OWNERWRITE and TPMA_NV_OWNERREAD) give the platform and owner access to NV Indexes using Platform Authorization or Owner Authorization rather than the *authValue* or *authPolicy* of the Index.

If access to an NV Index is to be restricted based on PCR, then an appropriate *authPolicy* shall be provided.

NOTE *platformAuth* or *ownerAuth* can be provided in any type of authorization session or as a password.

If TPMA_NV_AUTHREAD is SET, then the Index may be read if the Index *authValue* is provided. If TPMA_NV_POLICYREAD is SET, then the Index may be read if the Index *authPolicy* is satisfied.

At least one of TPMA_NV_PPREAD, TPMA_NV_OWNERREAD, TPMA_NV_AUTHREAD, or TPMA_NV_POLICYREAD shall be SET.

If TPMA_NV_AUTHWRITE is SET, then the Index may be written if the Index *authValue* is provided. If TPMA_NV_POLICYWRITE is SET, then the Index may be written if the Index *authPolicy* is satisfied.

At least one of TPMA_NV_PPWRITE, TPMA_NV_OWNERWRITE TPMA_NV_AUTHWRITE, or TPMA_NV_POLICYWRITE shall be SET.

If TPMA_NV_WRITELOCKED is SET, then the Index may not be written. If TPMA_NV_WRITEDEFINE is SET, TPMA_NV_WRITELOCKED may not be CLEAR except by deleting and redefining the Index. If TPMA_NV_WRITEDEFINE is CLEAR, then TPMA_NV_WRITELOCKED will be CLEAR on the next TPM2_Startup(TPM_SU_CLEAR).

NOTE If TPMA_NV_WRITELOCKED is SET, but TPMA_NV_WRITTEN is CLEAR, then TPMA_NV_WRITELOCKED is CLEAR by TPM Reset or TPM Restart. This action occurs even if the TPMA_NV_WRITEDEFINE attribute is SET. This action prevents an NV Index from being defined that can never be written, and permits a use case where an Index is defined, but the user wants to prohibit writes until after a reboot.

If TPMA_NV_READLOCKED is SET, then the Index may not be read. TPMA_NV_READLOCKED will be CLEAR on the next TPM2_Startup(TPM_SU_CLEAR).

NOTE The TPM is expected to maintain indicators to indicate that the Index is temporarily locked. The state of these indicators is reported in the TPMA_NV_READLOCKED and TPMA_NV_WRITELOCKED attributes.

If the TPM_NT is TPM_NT_EXTEND, then writes to the Index will cause an update of the Index using the extend operation with the *nameAlg* used to create the digest.

If TPM_NT is TPM_NT_PIN_FAIL, TPMA_NV_NO_DA must be SET. This removes ambiguity over which Dictionary Attack defense protects a TPM_NV_PIN_FAIL's *authValue*.

When the Index is created (TPM2_NV_DefineSpace()), TPMA_NV_WRITELOCKED, TPMA_NV_READLOCKED, and TPMA_NV_WRITTEN shall all be CLEAR in the parameter that defines the attributes of the created Index.

Table 204 — Definition of (UINT32) TPMA_NV Bits

Bit	Name	Description
0	TPMA_NV_PPWRITE	**SET (1):** The Index data can be written if Platform Authorization is provided. **CLEAR (0):** Writing of the Index data cannot be authorized with Platform Authorization.
1	TPMA_NV_OWNERWRITE	**SET (1):** The Index data can be written if Owner Authorization is provided. **CLEAR (0):** Writing of the Index data cannot be authorized with Owner Authorization.
2	TPMA_NV_AUTHWRITE	**SET (1):** Authorizations to change the Index contents that require USER role may be provided with an HMAC session or password. **CLEAR (0):** Authorizations to change the Index contents that require USER role may not be provided with an HMAC session or password.
3	TPMA_NV_POLICYWRITE	**SET (1):** Authorizations to change the Index contents that require USER role may be provided with a policy session. **CLEAR (0):** Authorizations to change the Index contents that require USER role may not be provided with a policy session. NOTE TPM2_NV_ChangeAuth() always requires that authorization be provided in a policy session.
7:4	TPM_NT	The type of the index. NOTE A TPM is not required to support all TPM_NT values
9:8	Reserved	shall be zero reserved for future use
10	TPMA_NV_POLICY_DELETE	**SET (1):** Index may not be deleted unless the *authPolicy* is satisfied using TPM2_NV_UndefineSpaceSpecial(). **CLEAR (0):** Index may be deleted with proper platform or owner authorization using TPM2_NV_UndefineSpace(). NOTE An Index with this attribute and a policy that cannot be satisfied (e.g., an Empty Policy) cannot be deleted.
11	TPMA_NV_WRITELOCKED	**SET (1):** Index cannot be written. **CLEAR (0):** Index can be written.
12	TPMA_NV_WRITEALL	**SET (1):** A partial write of the Index data is not allowed. The write size shall match the defined space size. **CLEAR (0):** Partial writes are allowed. This setting is required if the *.dataSize* of the Index is larger than NV_MAX_BUFFER_SIZE for the implementation.
13	TPMA_NV_WRITEDEFINE	**SET (1):** TPM2_NV_WriteLock() may be used to prevent further writes to this location. **CLEAR (0):** TPM2_NV_WriteLock() does not block subsequent writes if TPMA_NV_WRITE_STCLEAR is also CLEAR.
14	TPMA_NV_WRITE_STCLEAR	**SET (1):** TPM2_NV_WriteLock() may be used to prevent further writes to this location until the next TPM Reset or TPM Restart. **CLEAR (0):** TPM2_NV_WriteLock() does not block subsequent writes if TPMA_NV_WRITEDEFINE is also CLEAR.
15	TPMA_NV_GLOBALLOCK	**SET (1):** If TPM2_NV_GlobalWriteLock() is successful, then further writes to this location are not permitted until the next TPM Reset or TPM Restart. **CLEAR (0):** TPM2_NV_GlobalWriteLock() has no effect on the writing of the data at this Index.

Bit	Name	Description
16	TPMA_NV_PPREAD	**SET (1):** The Index data can be read if Platform Authorization is provided. **CLEAR (0):** Reading of the Index data cannot be authorized with Platform Authorization.
17	TPMA_NV_OWNERREAD	**SET (1):** The Index data can be read if Owner Authorization is provided. **CLEAR (0):** Reading of the Index data cannot be authorized with Owner Authorization.
18	TPMA_NV_AUTHREAD	**SET (1):** The Index data may be read if the *authValue* is provided. **CLEAR (0):** Reading of the Index data cannot be authorized with the Index *authValue*.
19	TPMA_NV_POLICYREAD	**SET (1):** The Index data may be read if the *authPolicy* is satisfied. **CLEAR (0):** Reading of the Index data cannot be authorized with the Index *authPolicy*.
24:20	Reserved	shall be zero reserved for future use
25	TPMA_NV_NO_DA	**SET (1):** Authorization failures of the Index do not affect the DA logic and authorization of the Index is not blocked when the TPM is in Lockout mode. **CLEAR (0):** Authorization failures of the Index will increment the authorization failure counter and authorizations of this Index are not allowed when the TPM is in Lockout mode.
26	TPMA_NV_ORDERLY	**SET (1):** NV Index state is only required to be saved when the TPM performs an orderly shutdown (TPM2_Shutdown()). **CLEAR (0):** NV Index state is required to be persistent after the command to update the Index completes successfully (that is, the NV update is synchronous with the update command). NOTE If TPMA_NV_ORDERLY is SET, TPMA_NV_WRITTEN will be CLEAR by TPM Reset.
27	TPMA_NV_CLEAR_STCLEAR	**SET (1):** TPMA_NV_WRITTEN for the Index is CLEAR by TPM Reset or TPM Restart. **CLEAR (0):** TPMA_NV_WRITTEN is not changed by TPM Restart. NOTE This attribute may only be SET if TPM_NT is not TPM_NT_COUNTER.
28	TPMA_NV_READLOCKED	**SET (1):** Reads of the Index are blocked until the next TPM Reset or TPM Restart. **CLEAR (0):** Reads of the Index are allowed if proper authorization is provided.
29	TPMA_NV_WRITTEN	**SET (1):** Index has been written. **CLEAR (0):** Index has not been written.
30	TPMA_NV_PLATFORMCREATE	**SET (1):** This Index may be undefined with Platform Authorization but not with Owner Authorization. **CLEAR (0):** This Index may be undefined using Owner Authorization but not with Platform Authorization. The TPM will validate that this attribute is SET when the Index is defined using Platform Authorization and will validate that this attribute is CLEAR when the Index is defined using Owner Authorization.
31	TPMA_NV_READ_STCLEAR	**SET (1):** TPM2_NV_ReadLock() may be used to SET TPMA_NV_READLOCKED for this Index. **CLEAR (0):** TPM2_NV_ReadLock() has no effect on this Index.

13.5 TPMS_NV_PUBLIC

This structure describes an NV Index.

Table 205 — Definition of TPMS_NV_PUBLIC Structure

Name	Type	Description
nvIndex	TPMI_RH_NV_INDEX	the handle of the data area
nameAlg	TPMI_ALG_HASH	hash algorithm used to compute the name of the Index and used for the *authPolicy.* For an extend index, the hash algorithm used for the extend.
attributes	TPMA_NV	the Index attributes
authPolicy	TPM2B_DIGEST	optional access policy for the Index The policy is computed using the *nameAlg* NOTE Shall be the Empty Policy if no authorization policy is present.
dataSize{:MAX_NV_INDEX_SIZE}	UINT16	the size of the data area The maximum size is implementation-dependent. The minimum maximum size is platform-specific.
#TPM_RC_SIZE		response code returned when the requested size is too large for the implementation

13.6 TPM2B_NV_PUBLIC

This structure is used when a TPMS_NV_PUBLIC is sent on the TPM interface.

Table 206 — Definition of TPM2B_NV_PUBLIC Structure

Name	Type	Description
size=	UINT16	size of *nvPublic*
nvPublic	TPMS_NV_PUBLIC	the public area

14 Context Data

14.1 Introduction

Clause 14 defines the contents of the TPM2_ContextSave() response parameters and TPM2_ContextLoad() command parameters.

If the parameters provided by the caller in TPM2_ContextLoad() do not match the values returned by the TPM when the context was saved, the integrity check of the TPM2B_CONTEXT will fail and the object or session will not be loaded.

14.2 TPM2B_CONTEXT_SENSITIVE

This structure holds the object or session context data. When saved, the full structure is encrypted.

NOTE This is an informative table that is included in the specification only to allow calculation of the maximum size for TPM2B_CONTEXT_DATA.

Table 207 — Definition of TPM2B_CONTEXT_SENSITIVE Structure <IN/OUT>

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

14.3 TPMS_CONTEXT_DATA

This structure holds the integrity value and the encrypted data for a context.

NOTE This is an informative table that is included in the specification only to allow calculation of the maximum size for TPM2B_CONTEXT_DATA.

Table 208 — Definition of TPMS_CONTEXT_DATA Structure <IN/OUT, S>

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

14.4 TPM2B_CONTEXT_DATA

This structure is used in a TPMS_CONTEXT.

Table 209 — Definition of TPM2B_CONTEXT_DATA Structure <IN/OUT>

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

14.5 TPMS_CONTEXT

This structure is used in TPM2_ContextLoad() and TPM2_ContextSave(). If the values of the TPMS_CONTEXT structure in TPM2_ContextLoad() are not the same as the values when the context was saved (TPM2_ContextSave()), then the TPM shall not load the context.

Saved object contexts shall not be loaded as long as the associated hierarchy is disabled.

Saved object contexts are invalidated when the Primary Seed of their hierarchy changes. Objects in the Endorsement hierarchy are invalidated when either the EPS or SPS is changed.

When an object has the *stClear* attribute, it shall not be possible to reload the context or any descendant object after a TPM Reset or TPM Restart.

NOTE 1 The reference implementation prevents reloads after TPM Restart by including the current value of a *clearCount* in the saved object context. When an object is loaded, this value is compared with the current value of the *clearCount* if the object has the *stClear* attribute. If the values are not the same, then the object cannot be loaded.

A sequence value is contained within *contextBlob*, the integrity-protected part of the saved context. The sequence value is repeated in the *sequence* parameter of the TPMS_CONTEXT structure. The *sequence* parameter, along with other values, is used in the generation the protection values of the context.

NOTE 2 The reference implementation prepends the *sequence* value to the *contextBlob* before, for example, the SESSION structure for sessions or the OBJECT structure for transient objects.

If the integrity value of the context is valid, but the *sequence* value of the decrypted context does not match the value in the *sequence* parameter, then TPM shall enter the failure mode because this is indicative of a specific type of attack on the context values.

NOTE 3 If the integrity value is correct, but the decryption fails and produces the wrong value for sequence, this implies that either the TPM is faulty or an external entity is able to forge an integrity value for the context but they have insufficient information to know the encryption key of the context. Since the TPM generated the valid context, then there is no reason for the sequence value in the context to be decrypted incorrectly other than the TPM is faulty or the TPM is under attack. In either case, it is appropriate for the TPM to enter failure more.

Table 210 — Definition of TPMS_CONTEXT Structure

Name	Type	Description
sequence	UINT64	the sequence number of the context NOTE Transient object contexts and session contexts used different counters.
savedHandle	TPMI_DH_CONTEXT	a handle indicating if the context is a session, object, or sequence object See Table 211 — Context Handle Values
hierarchy	TPMI_RH_HIERARCHY+	the hierarchy of the context
contextBlob	TPM2B_CONTEXT_DATA	the context data and integrity HMAC

14.6 Parameters of TPMS_CONTEXT

14.6.1 *sequence*

The *sequence* parameter is used to differentiate the contexts and to allow the TPM to create a different encryption key for each context. Objects and sessions use different sequence counters. The sequence counter for objects (transient and sequence) is incremented when an object context is saved, and the

sequence counter for sessions increments when a session is created or when it is loaded (TPM2_ContextLoad()). The session sequence number is the *contextID* counter.

For a session, the sequence number also allows the TRM to find the “older” contexts so that they may be refreshed if the *contextID* are too widely separated.

If an input value for *sequence* is larger than the value used in any saved context, the TPM shall return an error (TPM_RC_VALUE) and do no additional processing of the context.

If the context is a session context and the input value for sequence is less than the current value of *contextID* minus the maximum range for sessions, the TPM shall return an error (TPM_RC_VALUE) and do no additional processing of the context.

14.6.2 *savedHandle*

For a session, this is the handle that was assigned to the session when it was created. For a transient object, the handle will have one of the values shown in Table 211.

If the handle type for *savedHandle* is TPM_HT_TRANSIENT, then the low order bits are used to differentiate static objects from sequence objects.

If an input value for handle is outside of the range of values used by the TPM, the TPM shall return an error (TPM_RC_VALUE) and do no additional processing of the context.

Table 211 — Context Handle Values

Value	Description
0x02xxxxxx	an HMAC session context
0x03xxxxxx	a policy session context
0x80000000	an ordinary transient object
0x80000001	a sequence object
0x80000002	a transient object with the *stClear* attribute SET

14.6.3 *hierarchy*

This is the hierarchy (TPMI_RH_HIERARCHY) for the saved context and determines the proof value used in the construction of the encryption and integrity values for the context. For session and sequence contexts, the hierarchy is TPM_RC_NULL. The hierarchy for a transient object may be TPM_RH_NULL but it is not required.

14.7 Context Protection

14.7.1 Context Integrity

The integrity of the context blob is protected by an HMAC. The integrity value is constructed such that changes to the component values will invalidate the context and prevent it from being loaded.

Previously saved contexts for objects in the Platform hierarchy shall not be loadable after the PPS is changed.

Previously saved contexts for objects in the Storage hierarchy shall not be loadable after the SPS is changed.

Previously saved contexts for objects in the Endorsement hierarchy shall not be loadable after either the EPS or SPS is changed.

Previously saved sessions shall not be loadable after the SPS changes.

Previously saved contexts for objects that have their *stClear* attribute SET shall not be loadable after a TPM Restart. If a Storage Key has its *stClear* attribute SET, the descendants of this key shall not be loadable after TPM Restart.

Previously saved contexts for a session and objects shall not be loadable after a TPM Reset.

A saved context shall not be loaded if its HMAC is not valid. The equation for computing the HMAC for a context is found in “Context Integrity Protection” in TPM 2.0 Part 1.

14.7.2 Context Confidentiality

The context data of sessions and objects shall be protected by symmetric encryption using CFB. The method for computing the IV and encryption key is found in “Context Confidentiality Protection” in TPM 2.0 Part 1.

15 Creation Data

15.1 TPMS_CREATION_DATA

This structure provides information relating to the creation environment for the object. The creation data includes the parent Name, parent Qualified Name, and the digest of selected PCR. These values represent the environment in which the object was created. Creation data allows a relying party to determine if an object was created when some appropriate protections were present.

When the object is created, the structure shown in Table 212 is generated and a ticket is computed over this data.

If the parent is a permanent handle (TPM_RH_OWNER, TPM_RH_PLATFORM, TPM_RH_ENDORSEMENT, or TPM_RH_NULL), then *parentName* and *parentQualifiedName* will be set to the parent handle value and *parentNameAlg* will be TPM_ALG_NULL.

Table 212 — Definition of TPMS_CREATION_DATA Structure <OUT>

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

15.2 TPM2B_CREATION_DATA

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

Table 213 — Definition of TPM2B_CREATION_DATA Structure <OUT>

Parameter	Type	Description
size=	UINT16	size of the creation data
creationData	TPMS_CREATION_DATA	