

# Trusted Computing Platform Alliance (TCPA)

## *Main Specification Version 1.0*

Copyright © 2000 Compaq Computer Corporation, Hewlett-Packard Company, IBM Corporation,  
Intel Corporation, Microsoft Corporation

All rights reserved.

### DISCLAIMERS:

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE.

NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED OR INTENDED HEREBY.

COMPAQ, HP, IBM, INTEL, AND MICROSOFT, DISCLAIM ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF PROPRIETARY RIGHTS, RELATING TO THE USE OF THE INFORMATION IN THIS SPECIFICATION AND TO THE IMPLEMENTATION OF INFORMATION IN THIS SPECIFICATION. COMPAQ, HP, IBM, INTEL, AND MICROSOFT, DO NOT WARRANT OR REPRESENT THAT SUCH IMPLEMENTATION(S) WILL NOT INFRINGE SUCH RIGHTS.

WITHOUT LIMITATION, COMPAQ, HP, IBM, INTEL, AND MICROSOFT DISCLAIM ALL LIABILITY FOR COST OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES, LOST PROFITS, LOSS OF USE, LOSS OF DATA OR ANY INCIDENTAL, CONSEQUENTIAL, DIRECT, INDIRECT, OR SPECIAL DAMAGES, WHETHER UNDER CONTRACT, TORT, WARRANTY OR OTHERWISE, ARISING IN ANY WAY OUT OF USE OR RELIANCE UPON THIS SPECIFICATION OR ANY INFORMATION HEREIN.

All product names are trademarks, registered trademarks, or service marks of their respective owners.

# Acknowledgement

TCPA wishes to thank members of the PKI, PC Specific and Conformance Workgroup who contributed expertise and text to this document. Thanks must be given to the members of the TCPA Technical Committee who were Michael Angelo, Boris Balacheff, Josh Benaloh, David Challener, Dhruv Desai, Paul England, David Grawrock, Bob Meinschein, Manny Novoa, Graeme Proudler, Jim Ward and Monty Wiseman.

David Chan  
Technical Committee Chair

## Change History

Version	Date	Description
0.44	July 2000	Voted by members as appropriate for public release with modifications.
0.90	August 2000	First version released to public.
0.91	October 26, 2000	Remove chapters 1 & 2. Complete reformat
0.92	4 November, 2000	Added new chapter for structures, updated functions to match IDL, editing changes.
1.0 RC1	28 November 2000	Incorporated comments cleaned up structures and made ready for publication.
1.0 RC2	11 December 2000	Incorporated changes from reflector. Added new change authorization command.
1.0 RC4	10 Jan 2001	Incorporated changes and fixed up IDL
1.0 RC5	11 Jan 2001	PKCS#1 changes



## Table of Contents

1.	Forward .....	1
2.	The Trusted Platform Subsystem.....	2
2.1	Introduction .....	2
2.2	Roots of Trust .....	2
2.3	Integrity Operations .....	5
2.3.1	Storage of Integrity Metrics.....	5
2.3.2	Reporting of Integrity Metrics .....	5
2.4	Use of Keys Associated with TPM Identities .....	6
2.5	Cryptographic Operations .....	7
2.6	Opting to use a TPM .....	8
2.6.1	Enabling Ownership .....	9
2.6.2	Activating a TPM .....	9
2.6.3	Selected operations .....	10
3.	Protection.....	13
3.1	Introduction .....	13
3.2	Threat .....	13
3.3	Integrity.....	14
3.4	Privileged Access.....	14
3.5	Side effects.....	14
4.	Structures and Defines .....	15
4.1.1	Endness of Structures .....	15
4.1.2	Byte Packing .....	15
4.2	Defines .....	16
4.2.1	Basic data types .....	16
4.2.2	Helper redefinitions .....	16
4.2.3	Vendor specific.....	17
4.3	Return codes.....	18
4.4	IDL .....	20
4.5	TCPA_VERSION.....	21
4.6	TCPA_DIGEST.....	22
4.7	TCPA_NONCE .....	23
4.8	TCPA_AUTHDATA.....	24
4.9	TCPA_PAYLOAD_TYPE .....	24
4.10	TCPA_INTERNAL_HDR.....	25
4.11	TCPA_PUBKEY.....	26
4.11.1	TCPA_RSA_PUBKEY.....	26
4.12	TCPA_PRIVKEY .....	27
4.13	TCPA_KEY .....	28
4.14	TCPA_PCR_REGISTER.....	29
4.15	TCPA_PCR_EVENT .....	30
4.16	TCPA_AUDIT_EVENT structure .....	32
4.17	Storage Structures .....	33
4.17.1	TCPA_SEALED_DATA .....	33
4.17.2	TCPA_PCR_SELECTION .....	35
4.17.3	TCPA_PCR_COMPOSITE .....	36
4.17.4	TCPA_KEY_FLAGS .....	36
4.17.5	TCPA_ASYM_HASH .....	37
4.17.6	TCPA_STORE_ASYMKEY .....	38
4.17.7	TCPA_MIGRATE_ASYMKEY.....	41
4.17.8	TCPA_MAINTENANCE_ASYMKEY.....	42
4.18	TCPA_AUTH .....	43
4.19	TCPA_CERTIFY_INFO Structure.....	44
4.20	TCPA_QUOTE_INFO Structure.....	45
4.21	TCPA_KEY_INFO.....	46

4.22	Flag Structures .....	47
4.22.1	TCPA_PERSISTENT_FLAGS Structure.....	48
4.22.2	TCPA_VOLATILE_FLAGS Structure.....	50
4.23	Credentials .....	51
4.23.1	Evidence of Subsystem Endorsement .....	52
4.23.2	Evidence of Platform Endorsement .....	54
4.23.3	Evidence of Platform Conformance.....	56
4.23.4	TCPA Validation Data .....	58
4.23.5	Evidence of Trusted Platform Module Identity .....	59
4.24	TCPA_ALGORITHM_PARMS.....	61
4.25	Identity Structures .....	62
4.25.1	TCPA_IDENTITY_CONTENTS .....	62
4.25.2	TCPA_SYMMETRIC_KEY .....	63
4.25.3	TCPA_IDENTITY_REQ.....	64
4.25.4	TCPA_SYM_IDENTITY_REQ.....	65
4.25.5	TCPA_ASYM_IDENTITY_REQ .....	66
4.25.6	TCPA_ASYM_CA_CONTENTS.....	67
4.25.7	TCPA_SYM_CA_ATTESTATION .....	68
4.26	TCPA_CHANGEAUTH_VALIDATE .....	69
4.27	TCPA_MIGRATIONKEYAUTH .....	70
4.28	TCPA_PROTOCOL_ID .....	71
4.29	TCPA_ENTITY_TYPE.....	72
4.30	TCPA_STARTUP_TYPE .....	73
4.31	Command Ordinals .....	74
5.	Authorization and Ownership .....	76
5.1	Introduction .....	76
5.2	Authorization protocols .....	78
5.2.1	OI-AP description .....	79
5.2.2	TPM_OIAP.....	81
5.2.3	Authorization using an OI-AP session .....	82
5.2.4	OS-AP Description.....	84
5.2.5	TPM_OSAP .....	87
5.2.6	Authorization using an OS-AP session .....	88
5.3	TPM_Terminate_Handle .....	91
5.4	ADIP – Creating a New Entity.....	92
5.5	ADCP - Changing Authorization Data.....	94
5.6	TPM_ChangeAuth.....	95
5.7	Asymmetric Authorization Change Protocol .....	97
5.7.1	TPM_ChangeAuthAsymStart .....	97
5.7.2	TPM_ChangeAuthAsymFinish.....	100
5.8	Authorization Data.....	102
5.9	Nonces .....	102
5.10	Authorization Handle.....	103
5.11	HMAC Calculation.....	104
5.11.1	HMAC Long Parameters .....	105
5.12	TPM Ownership.....	106
5.12.1	TPM_TakeOwnership .....	107
6.	Integrity Collection and Reporting .....	109
6.1	Introduction .....	109
6.2	Platform Configuration Registers .....	110
6.2.1	Format and Properties .....	110
6.2.2	Initialization .....	110
6.2.3	Authorized PCRs .....	110
6.3	Operations Supporting Integrity Collection and Reporting .....	111
6.3.1	TPM_Extend .....	111
6.3.2	TPM_PcrRead .....	112

6.3.3	TPM_Quote.....	113
6.3.4	TSS_LogExtendEvent.....	115
6.3.5	TSS_GetExtendEvent.....	117
6.3.6	TSS_GetExtendEventLog.....	119
6.3.7	TSS_DisposeEventLog.....	120
6.3.8	TPM_DirWriteAuth.....	121
6.3.9	TPM_DirRead.....	122
7.	Protected Storage.....	123
7.1	Introduction.....	125
7.1.1	Characteristics.....	125
7.1.2	Key Storage.....	127
7.2	Mandatory Functions.....	127
7.2.1	TPM_Seal.....	128
7.2.2	TPM_Unseal.....	131
7.2.3	TSS_Bind.....	134
7.2.4	TPM_UnBind.....	136
7.2.5	TPM_CreateWrapKey.....	137
7.2.6	TPM_CreateWrapKeyToPcr.....	139
7.2.7	TSS_WrapKey.....	141
7.2.8	TSS_WrapKeyToPcr.....	143
7.2.9	TPM_LoadKey.....	145
7.2.10	TPM_GetPubKey.....	147
7.2.11	TPM_CreateMigrationBlob.....	148
7.2.12	TPM_MigrateMigrationBlob.....	150
7.2.13	TPM_LoadMigrationBlob.....	152
7.2.14	TPM_AuthorizeMigrationKey.....	154
7.3	TPM Optional Functions: Maintenance.....	155
7.3.1	TPM_CreateMaintenanceArchive.....	157
7.3.2	TPM_LoadMaintenanceArchive.....	159
7.3.3	TPM_KillMaintenanceFeature.....	161
8.	Cryptographic and Miscellaneous Functions.....	162
8.1	Introduction.....	162
8.2	Hash Operations.....	162
8.2.1	TSS_HashAll.....	163
8.2.2	TSS_HashInit.....	164
8.2.3	TSS_HashUpdate.....	165
8.2.4	TSS_HashFinal.....	166
8.3	HMAC Commands.....	167
8.3.1	TSS_HMACAll.....	168
8.3.2	TSS_HMACInit.....	169
8.3.3	TSS_HMACUpdate.....	170
8.3.4	TSS_HMACFinal.....	171
8.4	Key Certification.....	172
8.4.1	TPM_CertifyKey.....	172
8.5	Symmetric Encryption.....	174
8.5.1	TSS_EncryptAll.....	175
8.5.2	TSS_EncryptInit.....	176
8.5.3	TSS_EncryptUpdate.....	177
8.5.4	TSS_EncryptFinal.....	178
8.5.5	TSS_DecryptAll.....	179
8.5.6	TSS_DecryptInit.....	180
8.5.7	TSS_DecryptUpdate.....	181
8.5.8	TSS_DecryptFinal.....	182
8.6	Digital Signatures.....	183
8.6.1	TPM_Sign.....	183
8.6.2	TSS_VerifySignature.....	184

8.7	Random Numbers .....	185
8.7.1	TPM_GetRandom.....	186
8.7.2	TPM_StirRandom .....	187
8.8	Self Test .....	188
8.8.1	TPM_SelfTestFull .....	189
8.8.2	TPM_SelfTestStartup.....	190
8.8.3	TPM_CertifySelfTest.....	191
8.9	Reset and Clear Operations .....	192
8.9.1	TPM_Reset .....	193
8.9.2	TPM_Init .....	194
8.9.3	TPM_SaveState .....	195
8.9.4	TPM_Startup.....	196
8.9.5	TPM_OwnerClear.....	197
8.9.6	TPM_DisableOwnerClear .....	198
8.9.7	TPM_ForceClear .....	199
8.9.8	TPM_DisableForceClear .....	200
8.10	The GetCapability Commands .....	201
8.10.1	TPM_GetCapability.....	202
8.10.2	TSS_GetCapability .....	204
8.10.3	TPM_GetCapabilitySigned .....	205
8.11	Audit Commands .....	207
8.11.1	TPM_GetAuditEvent .....	208
8.11.2	TSS_GetAuditLog.....	209
8.11.3	TPM_SetOrdinalAuditStatus .....	210
8.11.4	TPM_GetOrdinalAuditStatus.....	211
8.12	Enabling Ownership .....	212
8.12.1	TPM_SetOwnerInstall .....	213
8.13	Enabling a TPM .....	214
8.13.1	TPM_OwnerSetDisable .....	215
8.13.2	TPM_PhysicalDisable .....	216
8.13.3	TPM_PhysicalEnable.....	217
8.14	Activating a TPM.....	218
8.14.1	TPM_PhysicalSetDeactivated.....	219
8.14.2	TPM_SetTempDeactivated.....	220
8.15	TPM_FieldUpgrade.....	221
8.16	TPM Internal RSA Operations on Arbitrarily Sized Data.....	223
8.16.1	TPM_Internal_Encrypt .....	224
8.16.2	TPM_Internal_Signature .....	226
8.17	TPM_SetRedirection .....	227
9.	Subsystem Credentials .....	229
9.1	Introduction.....	229
9.2	Endorsement.....	229
9.2.1	TPM_CreateEndorsementKeyPair .....	230
9.2.2	TPM_ReadPubek .....	232
9.2.3	TPM_DisablePubekRead .....	233
9.2.4	TPM_OwnerReadPubek .....	234
9.3	Generating a Trusted Platform Module Identity.....	235
9.3.1	TPM_MakeIdentity.....	238
9.3.2	TSS_CollateIdentityRequest.....	241
9.3.3	Contacting a Privacy CA .....	244
9.3.4	TPM_ActivateTPMIdentity .....	245
9.3.5	TSS_RecoverTPMIdentity .....	246
9.4	Instantiation of Data When Contacting a Privacy CA .....	248
9.4.1	From Owner to Privacy CA .....	248
9.4.2	From Privacy CA to Owner.....	249
9.5	Instantiation of Credentials as Certificates .....	251



9.5.1	Instantiation of TPM_ENDORSEMENT_CREDENTIALs .....	252
9.5.2	Instantiation of Platform_credentials .....	255
9.5.3	Instantiation of TPM_CONFORMANCE_CREDENTIAL .....	258
9.5.4	Instantiation of Validation Certificate .....	261
9.5.5	Instantiation of TPM_IDENTITY_CREDENTIAL .....	264
9.5.6	ASN.1 Definitions .....	268
10.	Conformance Criteria .....	270
10.1	Base Levels for Interoperability .....	270
10.2	Conformance Specification Sheet .....	270
10.3	Protocol Negotiation and Algorithm Agility .....	270
10.4	Cryptographic Algorithms and Protocols .....	271
10.4.1	Asymmetric .....	271
10.4.2	Symmetric .....	272
10.4.3	Hashing .....	272
10.4.4	Signature Operations .....	272
10.4.5	Creating a PCR composite hash .....	273
10.4.6	Using Secret Keys .....	273
10.5	Random Number Generator (RNG) .....	274
10.5.1	Entropy Source and Collector .....	274
10.5.2	State Register .....	274
10.5.3	Mixing Function .....	275
10.5.4	RNG Reset .....	275
10.6	Key Generation .....	275
10.6.1	Asymmetric .....	276
10.6.2	Symmetric .....	276
10.6.3	Nonce Creation .....	276
10.7	Auditing .....	276
10.8	Self-Tests .....	276
10.8.1	Required Self-Tests .....	277
10.8.2	Recommended Checks .....	277
10.8.3	Self-Test Failure .....	277
10.9	Object Reuse .....	277
10.10	Maintenance .....	278
10.11	Backup .....	278
10.12	Strength of Function .....	278
10.13	Protection Profile .....	279
10.14	Compliance to Specification .....	279
10.15	Field Upgrade .....	280
10.16	Physical Presence or Access .....	280
10.17	Other Specifications .....	281
11.	Appendix A: Glossary .....	282

# 1. Forward

This document is an industry specification that enables trust in computing platforms in general.

This specification defines a trusted *Subsystem* that is an integral part of each platform, and provides functions that can be used by enhanced operating systems and applications. The Subsystem employs cryptographic methods when establishing trust, and while this does not in itself convert a platform into a secure computing environment, it is a significant step in that direction.

Standardization is necessary so that the security and cryptographic community can assess the mechanisms involved, and so that customers can understand and trust the effectiveness of new features. Manufacturers will compete in the marketplace by installing Subsystems with varying capabilities and cost points. The Subsystem itself will have basic functions that maintain privacy, yet support the identity and authentication of entities such as the platform, the user, and other entities. The Subsystem will have other capabilities to protect data and verify certain operational aspects of the platform. It can be a separate device or devices, or it can be integrated into some existing component or components provided the implementation meets the requirements of this specification. This is necessary to achieve the fundamental goal of ubiquity.

Please note a very important distinction between different sections of text throughout this document. Beginning in chapter 2, "The Trusted Platform Subsystem," you will encounter two distinctive kinds of text: *informative comment* and *normative statements*. Because most of the text in this specification will be of the kind *normative statements*, the authors have informally defined it as the default and, as such, have specifically called out text of the kind *informative comment*. They have done this by flagging the beginning and end of each *informative comment* and highlighting its text in gray. This means that unless text is specifically marked as of the kind *informative comment*, you can consider it of the kind *normative statements*.

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in the chapters 2-10 normative statements are to be interpreted as described in [RFC-2119].

For example:

This is the first paragraph of 1–n paragraphs containing text of the kind *informative comment* ...

This is the second paragraph of text of the kind *informative comment* ...

This is the nth paragraph of text of the kind *informative comment* ...

To understand the TCPA specification the user must read the specification. (This use of MUST does not require any action).

This is the first paragraph of one or more paragraphs (and/or sections) containing the text of the kind *normative statements* ...

To understand the TCPA specification the user MUST read the specification. (This use of MUST indicates a keyword usage and requires an action).

## 2. The Trusted Platform Subsystem

### 2.1 Introduction

***Start of informative comment:***

The TCPA Subsystem design is to provide useful trust and security capabilities while minimizing the number of functions that must be trusted. This arrangement is necessary to make the Subsystem useful while remaining low in cost and can result in unusual features as compared with a conventional crypto co-processor.

***End of informative comment.***

### 2.2 Roots of Trust

***Start of informative comment:***

This section introduces the architectural aspects of a Trusted Platform that enable the collection and reporting of integrity metrics.

Among other things, a Trusted Platform enables an entity to determine the state of the software environment in that platform and to SEAL data to a particular software environment in that platform.

The entity deduces whether the state of the computing environment in that platform is acceptable and performs some transaction with that platform. If that transaction involves sensitive data that must be stored on the platform, the entity can ensure that that data is held in a confidential format unless the state of the computing environment in that platform is acceptable to the entity.

To enable this, a Trusted Platform provides information to enable the entity to deduce the software environment in a Trusted Platform. That information is reliably measured and reported to the entity. At the same time, a Trusted Platform provides a means to encrypt cryptographic keys and to state the software environment that must be in place before the keys can be decrypted.

Both these functions require integrity metrics. These metrics consist of data reflecting the integrity of the software state of the Trusted Platform. Both functions require two roots of trust in a platform. One is known as the “root of trust for measuring integrity metrics,” and the other is known as storing and reporting integrity metrics.”

The root of trust for measuring integrity metrics is likely to be different for different types of platforms because the metrics and their measurements will depend on the type of platform. The root of trust for storing and reporting integrity metrics enables integrity metrics to be reliably stored and reported and can have the same capabilities, irrespective of the type of platform.

A “trusted measurement root” measures certain platform characteristics, logs the measurement data in a measurement store, and stores the final result in a TPM (which contains the root of trust for storing and reporting integrity metrics). The trusted measurement root might also measure the characteristics of another measurement agent before passing control to the second agent. That second agent might repeat the process of measuring platform characteristics, storing measurement data and the final result, passing control to a third measurement agent, and so on.

When an integrity challenge is received, the Trusted Platform Agent gathers the following:

- the final results from the TPM,
- the log of the measurement data from the Trusted Platform Measurement Store, and
- TCPA Validation Data that states the values that the measurements should produce in a platform that is working correctly.

The Trusted Platform Agent then sends this measurement data to the Challenger. The Challenger uses the data to check that it is consistent with the final results and then compares the data (and perhaps the final results) with the TCPA Validation Data. This comparison enables the Challenger to deduce the

software state of the Trusted Platform and consequently decide whether the Challenger is satisfied to trust the platform for the intended purpose.

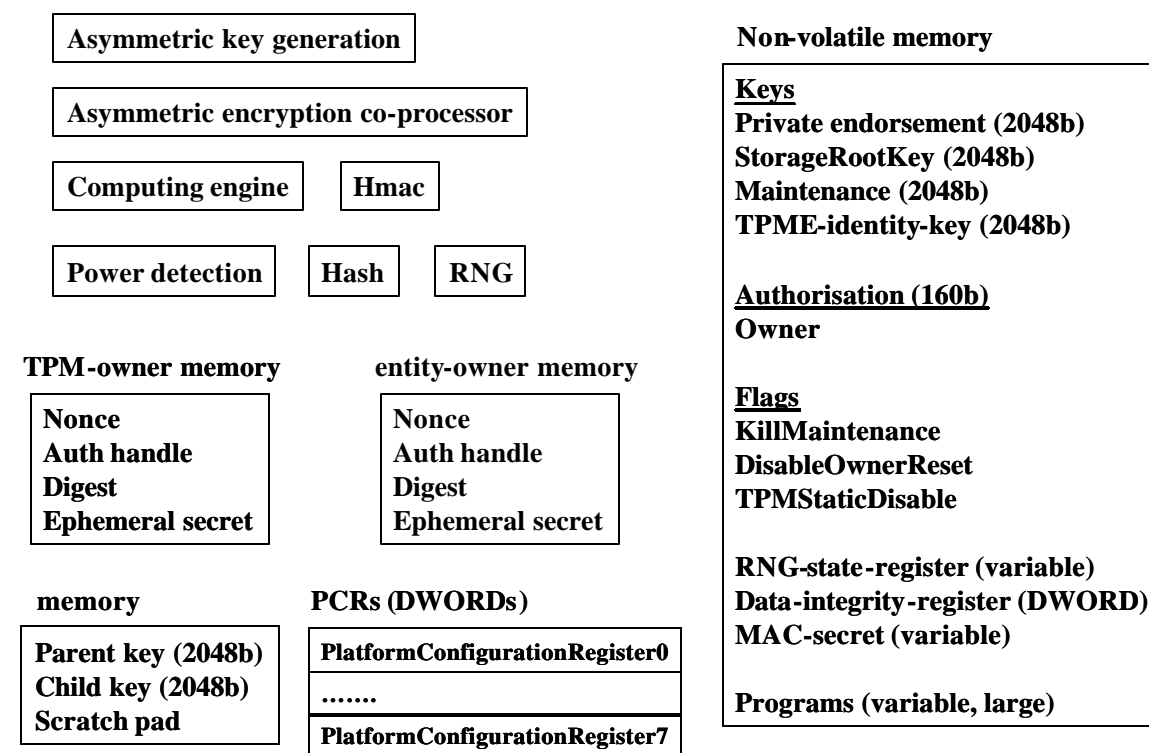
Once the Challenger has determined that the Trusted Platform can be trusted, the Challenger can use the TPM to store keys alongside stated values of integrity metrics, such that the TPM will not release the keys unless the current measured values of integrity metric match the stated values of integrity metric.

Both roots of trust, plus certain other capabilities for other purposes, must be implemented in ways that enable confidence in their correct operation in all circumstances of interest. A Challenger must be able to trust the roots and these capabilities. The implementation of the root of trust for measurement will typically vary depending on the type of platform (for example, PC, server, or phone). The TPM is defined as the set of all trusted capabilities apart from the root of trust for measurement, because these are independent of the type of platform. The whole Subsystem, therefore, typically consists of a root of trust for measuring integrity metrics, plus a TPM, plus other functions (the Support Services, or SS) that do not have to be trusted to function properly. Those other functions must still operate properly if the Subsystem is to operate properly, but any misbehavior of the SS can be detected. Any misbehavior of the functions in a root, or in the TPM, on the other hand, cannot be detected.

It is not the intention of this specification to specify the method of construction of either the Subsystem or the TPM, provided that they meet the requirements of this specification. The following diagram is an indication of the functional elements of a typical TPM.

**End of informative comment.**

## TPM contents



A Trusted Platform SHALL include the following:

- at least one root of trust for measuring integrity metrics,
- exactly one root of trust for storing and reporting integrity metrics,
- at least one Trusted Platform Measurement Store,

- at least one TCPA Validation Data, and
- exactly one Trusted Platform Agent.

The Endorsement Key is transitively bound to the Platform via the TPM as follows:

1. An Endorsement Key is bound to one and only one TPM (i.e., there is a one to one correspondence between an Endorsement Key and a TPM.)
2. A TPM is bound to one and only one Platform. (i.e., there is a one to one correspondence between a TPM and a Platform.)
3. Therefore, an Endorsement Key is bound to a Platform. (i.e., there is a one to one correspondence between an Endorsement Key and a Platform.)

An instantiation of the root of trust for measuring integrity metrics, while acting as the root of trust for measuring integrity metrics, SHALL do the following:

- execute no programs other than those intended by the entity that vouches for the root of trust for measuring integrity metrics,
- be resistant to the forms of software attack and to the forms of physical attack implied by the platform's Protection Profile,
- accurately measure at least one integrity metric that indicates the software environment of a platform,
- accurately record measured integrity metrics to a root of trust for storing and reporting integrity metrics, and
- accurately record details of the process of measuring all its integrity metrics to a Trusted Platform Measurement Store.

An instantiation of the root of trust for storing and reporting integrity metrics SHALL do the following:

- be resistant to all forms of software attack and to the forms of physical attack implied by the platform's Protection Profile,
- accept recording of measured integrity metrics, and
- supply an accurate digest of all sequences of presented integrity metrics.

An instantiation of a Trusted Platform Measurement Store SHOULD do the following:

- accurately accept, store and supply details of at least one process of measuring an integrity metric.

An instantiation of the repository for TCPA Validation Data SHOULD do the following:

- accurately store and supply a predicted value of at least one integrity metric.

An instantiation of the Trusted Platform Agent SHOULD do the following:

- obtain and supply an accurate report from the root of trust for storing and reporting integrity metrics of at least one sequence of integrity metrics in a form that prevents misrepresentation of that sequence or its source,
- obtain and supply an accurate report from a Trusted Platform Measurement Store of at least one set of details describing the measurement of an integrity metric, and

## 2.3 Integrity Operations

### 2.3.1 Storage of Integrity Metrics

***Start of informative comment:***

This section introduces the way that sequences of values of integrity metrics are stored in a TPM. This section does not describe the way that logs of the measurement process are stored in the Trusted Platform Measurement Store.

Each entry in the log inside the Trusted Platform Measurement Store contains a description of a measured entity plus an appropriate integrity metric that has been recorded inside a TPM. The log can be used to reproduce the value of each sequence of integrity metrics inside the TPM. If the log and the TPM are consistent and the TPM is trustworthy, the log can be trusted. If the values derived from the log and the values reported by the TPM are the same, the log is presumed to be an accurate record of the steps involved in building the software environment of the target platform. Consequently, the descriptions in the log of the measured entities represent the actual entities that contributed to the software environment inside the platform. Any difference between the values derived from the log and the values reported by the TPM indicate an undesirable inconsistency in the state of the target platform.

The mechanism used by the TPM to store sequences of values of integrity metrics is the subject of this section. This method must be reproduced when verifying the consistency of the values derived from the log and the values reported by the TPM.

A large number of integrity metrics may be measured in a platform, and a particular integrity metric may change with time and a new value may need to be stored. It is difficult to authenticate the source of measurement of integrity metrics, and as a result a new value of an integrity metric cannot be permitted to simply overwrite an existing value. (A rogue could erase an existing value that indicates subversion and replace it with a benign value.) Thus, if values of integrity metrics are individually stored, and updates of integrity metrics must be individually stored, it is difficult to place an upper bound on the size of memory that is required to store integrity metrics.

The TCPA solution is not to store individual integrity metrics. Instead, a Trusted Platform provides a way to store sequences of integrity metrics. Values of integrity metrics cannot be “stored” inside a TPM, and must instead be appended to a sequence. The states of all sequences inside a TPM are set to a known value at power-up. Each new integrity metric must be appended to a sequence and must modify the value of that sequence. The actual TCPA method is to concatenate the value of a new integrity metric with the existing value of the sequence, compute a digest of the concatenation, and use that digest as the new representation of the sequence.

This method enables one or more sequences to represent an arbitrary number of integrity metrics and their updates. The fewer the number of sequences, the more difficult it becomes to interpret the meaning of the value of a sequence. The greater the number of sequences, the more costly it becomes to provide storage. A particular implementation must make a trade-off between cost and difficulty of interpretation.

***End of informative comment.***

Integrity metrics that are presented to a TPM SHALL be stored inside that TPM in a way that prevents misrepresentation of the presented values or of the sequence in which they were presented.

### 2.3.2 Reporting of Integrity Metrics

***Start of informative comment:***

This section introduces the way that sequences of integrity metrics are reported by a TPM.

An entity seeking to know the state of the computing environment inside a Trusted Platform depends critically on the values of the integrity metrics. The integrity metrics enable an entity to determine the consistency of the measurement information and compare the actual and expected states of the platform.

It follows, then, that the integrity metrics must be reported by a trusted mechanism. That trusted mechanism is the TPM (which includes the root of trust for storing and reporting integrity metrics). The TPM proclaims its trustworthiness by signing data, using one of its identities and conventional cryptographic techniques. The signature key is known only to the TPM and is the private key of a key pair. The corresponding public key is an identity key, since it is a cryptographic value by which the TPM is known. Together, the signature key and the identity key are part of an identity of the TPM.

A person or (more probably) an organization vouches for the TPM by attesting to a TPM identity. Before agreeing to provide attestation, the organization checks the construction credentials of the TPM, the design credentials of the platform that incorporates the TPM, and the construction credentials of the platform that incorporates the TPM. When the TPM reports the values of the sequences of integrity metrics that it has stored, the TPM signs those values using a TPM identity. When an entity receives signed data that originated in a TPM, the entity can verify that the data has not been changed in transit. The entity can also check that the data was signed by a TPM identity and that an organization known to the entity has attested to the TPM identity.

The TPM uses a conventional method to defeat replay attacks. That is, the entity provides a nonce that the TPM concatenates with the sequence values, before signing the values, and the signed result is returned by the Trusted Platform Agent to the entity. The actual capability provided by the TPM may be considered to be an “integrity signature.” The TPM accepts arbitrary data, concatenates that arbitrary data with the sequence values, and signs the concatenated data using the signature key of a TPM identity. When providing sequence values, that arbitrary data is simply a nonce that was provided by the challenging entity. The signed data proves that the sequence values have been supplied by a “live” TPM.

At other times, the challenging entity may wish to obtain specific information from a Trusted Platform. Then, the arbitrary data could be a digest of the specific information. The signed data proves the state of the computing environment inside the Trusted Platform at the time that the specific information was supplied.

***End of informative comment.***

Sequences of integrity metrics reported by the TPM SHALL be reported by that TPM in a way that prevents misrepresentation of the sequences and prevents misrepresentation of the reporting TPM

## **2.4 Use of Keys Associated with TPM Identities**

***Start of informative comment:***

The private (signature) key associated with a TPM identity must be used only for signatures. (It is poor security practice to use the same asymmetric key for both signing and confidentiality.) If a TPM identity requires the use of confidentiality, the TPM must create a separate confidentiality key. A TPM identity can indicate that a confidentiality key “belongs” to a TPM identity by signing the confidentiality key.

The private (signature) keys associated with TPM identities must be used only for special operations and must be indelibly stored with flags that mark them as belonging to TPM identities. Currently, the special operations are signing sequence values, signing other keys that were generated inside the TPM, and signing data when obtaining attestation to the identity.

A TPM must use private keys associated with TPM identities only for these special purposes, and must refuse to use private keys associated with TPM identities for other purposes. Otherwise, a rogue may construct data (outside the TPM) that has the same format as that used by the TPM for these special operations, and cause a TPM to sign that data using a private key associated with TPM identity. Such data would be misinterpreted as genuine data constructed by the TPM for those special purposes, and could subvert the trust in those special purposes.

If the TPM prevents such a masquerade, a third party can always be certain that data (signed by a private key associated with a TPM identity) was actually generated by a TPM for one of those special operations. To avoid any possibility of confusion over which legitimate capability is using a TPM identity, any capability that signs using a TPM identity will perform the signature over data that includes the ordinal (label) of the command.

**End of informative comment.**

It MUST be possible to reliably distinguish between the private key of a TPM identity and other keys.

A key that is distinguished as the private key of a TPM identity SHALL NOT be used except for generating a digital signature value when the data being signed includes an accurate indication of the capability being executed. A TPM SHALL NOT use a key that is distinguished as the private key of a TPM identity except when signing on behalf of a TPM identity during the part of a TCPA “protected capability” whose specification requires the signature of a TPM identity.

When signing on behalf of a TPM identity during the part of a TCPA protected capability whose specification requires the signature of a TPM identity, a TPM SHALL NOT use a key other than one that is distinguished as the private key of a TPM identity.

## 2.5 Cryptographic Operations

**Start of informative comment:**

This section introduces the use of cryptographic operations within the Subsystem. Note that this specification does not include the AES. It is probable, however, that future versions of this specification will include the AES.

The Subsystem employs conventional cryptographic operations in conventional ways. Those operations include the following:

- Hashing (SHA-1)
- Random number generation (RNG)
- Asymmetric key generation (RSA)
- Asymmetric encryption/decryption (RSA)
- Symmetric encryption/decryption (3DES)

The Subsystem uses these capabilities to perform generation of random data, generation of asymmetric and symmetric keys, signing and confidentiality of stored data. The Subsystem also uses confidential messaging for its own purposes, but does not provide a general-purpose symmetric confidentiality service. This choice is deliberate, because the fundamental TCPA objective is to improve trust in a general-purpose computing platform. Hence, TCPA provides only those functions that are necessary to improve confidence in such a platform so that processing (including conventional security functions) on the platform can be done with greater confidence.

The TPM contains the minimum set of capabilities that are required to be trusted. The TPM capabilities must be trustworthy if the Subsystem is to be trusted. Other Subsystem capabilities must (of course) function properly if the Subsystem is to work as expected.

The TPM contains the following crypto capabilities:

- Hashing (SHA-1)
- Random number generation (RNG)
- Asymmetric key generation (RSA)
- Asymmetric encryption/decryption (RSA)

Note that this list does not include symmetric encryption. This is for reasons of cost.

The hash capability is for use primarily by the TPM, since the TPM requires access to a trusted hash function. The hash capability is exported by the TPM just to improve hash availability during the boot phase of a platform, when the “RTM” and other measurement agents probably have restricted access to the platform’s main processing engine.



The untrusted part of the Subsystem must include symmetric encryption functionality, but does not include an RNG. The TSS may also include duplicate asymmetric key generation and asymmetric encryption capabilities depending on the usefulness of TCPA protected capabilities to the TSS.

The Random Number Generator consists of a state-machine that accepts and mixes unpredictable data and a post-processor that is a one-way function (such as a hash algorithm). This architecture is chosen to provide a good source of random data without requiring that the TPM include a genuine source of unpredictable data (which may be expensive).

The state-machine has non-volatile state, is initialized with unpredictable data before delivery to a customer, and can at any time accept further (unpredictable) data. Such data may be provided by hardware (from thermal noise, for example), or by software (monitoring keyboard strokes, for example). Some such unpredictable data must be inserted every time that a platform boots. Naturally, a hardware source is likely to supply data at a higher baud rate than a software source. That “further data” is mixed into the existing state of the machine and as a result improves the unpredictability of the state of the state-machine. Neither the Owner of the TPM nor the manufacturer of the TPM can deduce the state of the state-machine. The post-processor is used to “condense” the output of the state-machine into data that has sufficient and uniform entropy. (The one-way function will use more bits of input data than it produces as output.)

***End of informative comment.***

## 2.6 Opting to use a TPM

***Start of informative comment:***

It is necessary to provide features that activate a TPM. This is for reasons of privacy.

A TPM is necessarily activated by a reset. This, however, causes the TPM to discard any existing secrets, and puts the TPM into its virgin state, waiting for an Owner. It leaves the TPM vulnerable to ownership by anyone who knows the PUBEK of the TPM and can get a “take ownership” command to the TPM. To fail safe, the true Owner would need to take ownership as soon as possible after a TPM has been reset. If desired, the true Owner could then withhold the authorization information that is necessary to use the TPM. Since a TPM can have only one Owner, this prevents any use of the TPM until the true Owner decides to use it.

It is therefore desirable to provide methods that deactivate and activate a TPM without destroying existing secrets. Then the Owner of the TPM (or a user) may deactivate the TPM in order to prevent inadvertent use of the TPM, and later reactivate the TPM in order to use current secrets. It is also desirable to provide methods that activate and deactivate the process of taking ownership, in case the true Owner does not wish to take ownership (at least, not yet).

The TCPA specification defines a set of capabilities to enable/disable a TPM, activate/deactivate a TPM, and enable/disable the process of taking ownership of the TPM.

The overall effect of the disabling capabilities is that a disabled TPM does little of value, apart from keeping accurate records of integrity metrics and acknowledging that the TPM exists. A disabled TPM is, therefore, effectively “off”.

The overall effect of the deactivating capabilities is that an inactive TPM does nothing, apart from keeping accurate records of integrity metrics, acknowledging that the TPM exists, and permitting the process of installing an owner in the TPM.

There are obviously many combinations of the particular states of TPM enabled/disabled, TPM active/inactive, install-owner enabled/disabled. It may be that some suppliers will choose to supply a virgin TPM that is enabled, active, and with “install owner” enabled, because that is what is required by their customer. At the other extreme, if a virgin TPM is supplied in the disabled and inactive state, with “take ownership” disabled, three steps are required in order to activate the TPM. One possible activation sequence would be:

1. The prospective Owner should enable the TPM.

2. The prospective Owner should attempt to take ownership.
3. The prospective Owner should activate the TPM.

This particular sequence gives maximum control to the Owner, and permits verification that taking ownership has succeeded, before the TPM is activated.

There are other possibilities between these two extremes. It may be that a virgin TPM is enabled but inactive, with “take ownership” disabled, for example. This may be an advantage if the process of enabling a TPM is non-trivial.

***End of informative comment.***

### 2.6.1 Enabling Ownership

***Start of informative comment:***

If a TPM does not have an Owner, it is desirable to provide a method that enables or disables the process by which a prospective Owner takes ownership of a TPM. Ideally this method would work both locally and remotely. Unfortunately authenticated commands cannot be interpreted by the TPM if it does not have an Owner. Hence the method of enabling or disabling the process of taking ownership is a local command, and no remote option is provided. (In a PC, these local controls could be made available during the POST, for example.)

***End of informative comment.***

### 2.6.2 Activating a TPM

***Start of informative comment:***

It is desirable to provide methods that activate or deactivate a TPM without permanently preventing access to secrets protected by the TPM. The provision of deactivation methods exposes a denial-of-service attack, but this is considered a worthwhile price to pay for improved privacy.

One method should certainly be the use of commands authorized by the Owner. This method has the advantage that it proves possession of sufficient privilege, and can be used either locally or remotely. A drawback of this method is that the platform must (probably) be fully active in order to communicate an authorized command to a TPM. The concern is that the TPM may inadvertently be used inbetween the platform becoming fully active and an authorized “deactivate” command being received by the TPM. Another disadvantage is that it may be necessary to disable a TPM when the Owner is not available. Other methods are, therefore, also required. The scope of these methods must reflect any uncertainty about possession of sufficient privilege.

One method is required to operate before the platform is fully active. In these circumstances, it may be difficult to check authorization. The method adopted by TCPA is to use software controls that are remotely inaccessible. These are intended to provide local activation only (not remote activation), but this depends upon the degree to which the control software is actually inaccessible to remote entities.

Another method is to required to operate when the platform is fully active, but without Owner authorization. The method adopted by the TCPA is to use an unauthorized command that has a limited effect – it can be used just to deactivate a TPM, and the effect lasts only until the platform is rebooted.

The method of final resort to activate a TPM is to use a physical (electrical) input to the TPM that cannot be controlled by software executing on the main platform. This method (obviously) provides local activation but not remote activation. This method is useful if no one has taken ownership, or the Owner's authorization has been lost, but one or more User authorization data are still known. In the latter case, the TPM can be activated and Users can use their secrets to recover as much as possible of their data.

This specification uses four methods of activation (while retaining current TPM secrets):

1. A physical (electrical) input to the TPM that cannot be controlled by software executing on the main platform. Enabling this physical input could involve opening of the platform and throwing a switch, or activation of a physical lock, for example. Each use of the control causes a transitory activate event at the TPM. This (obviously) provides local activation but not remote activation.
2. An authenticated command to the TPM from the Owner. This provides either local or remote activation of the TPM.
3. The use of software controls that are remotely inaccessible. These are intended to provide local activation and not remote activation, but that property depends upon the degree to which the controlling software is actually inaccessible to remote entities. (In a PC, these controls could be made available during the POST, for example.)
4. A power-cycle of the platform. This is intended to provide local activation and not remote activation, but that property depends upon the degree to which a reboot is actually inaccessible to remote entities.

This specification uses three methods of deactivation (while retaining current TPM secrets):

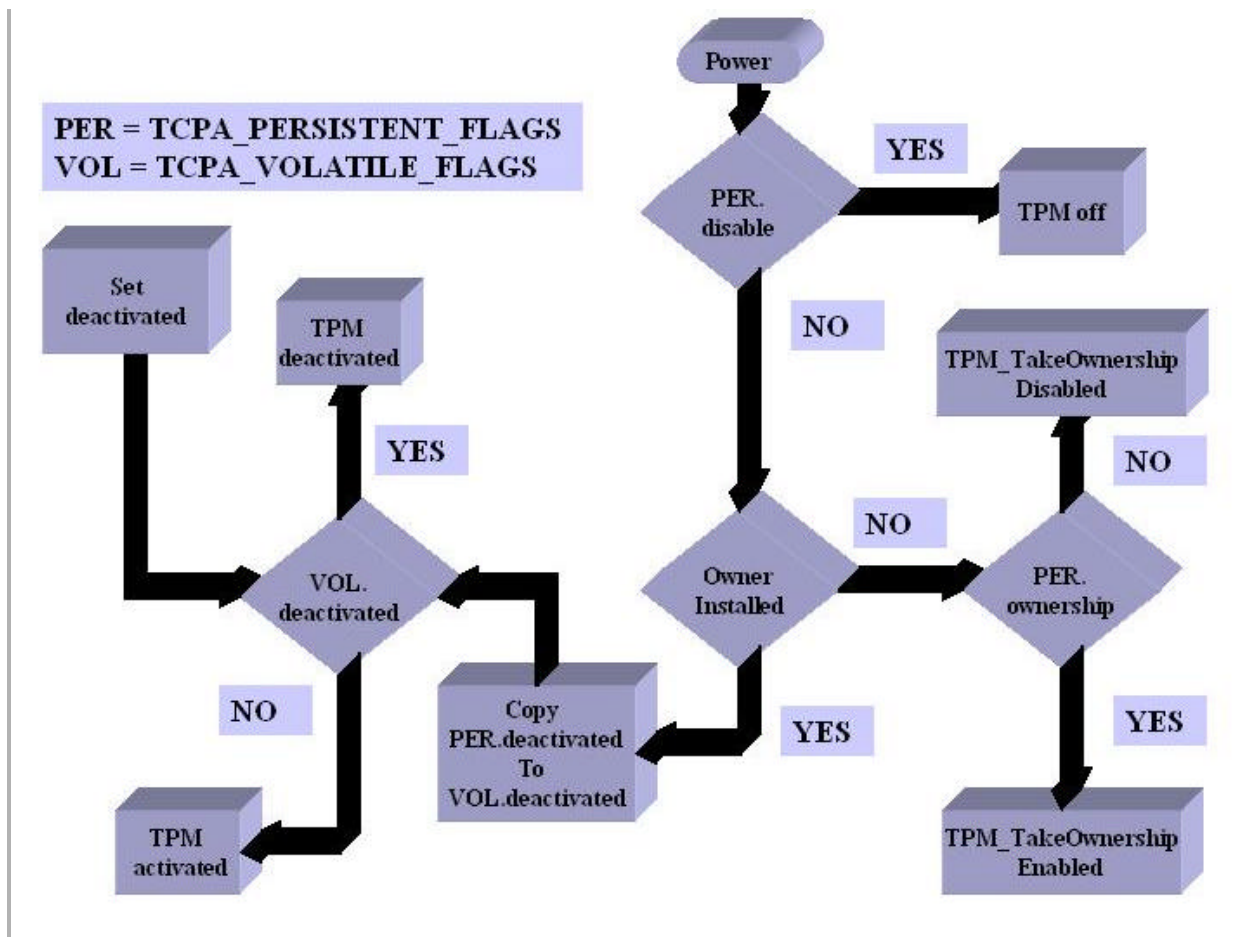
1. An authenticated command to the TPM from the Owner. These provide either local or remote deactivation of the TPM.
2. An unauthenticated command to the TPM. These provide either local or remote deactivation of the TPM.
3. The use of software controls that are remotely inaccessible. These are intended to provide local deactivation and not remote deactivation, but that property depends upon the degree to which the controlling software is actually inaccessible to remote entities. (In a PC, these controls could be made available during the POST, for example.)

***End of informative comment.***

### 2.6.3 Selected operations

***Start of informative comment:***

The methods to enable/disable a TPM, activate/deactivate a TPM, and enable/disable the process of taking ownership of the TPM, can be combined in many ways. The selection made by TCPA is illustrated in the following flowchart diagram, which illustrates a sequence of tests and decisions after Power-On-Reset (POR):



Bit	Flag name	Flag type	Action to set TRUE	Action to set FALSE
1	DISABLED_TPM	Non-volatile	1) Owner auth cmd 2) Local cmd	1) Owner auth cmd 2) physical action
2	DISABLED_OWNER_INSTALL	Non-volatile	Local cmd	Local cmd
3	DEACTIVATED_TPM	Non-volatile	Local cmd	Local cmd
4	TEMP_DEACTIVATED_TPM	Volatile	Unauth cmd	Platform reboot

(BIT1) This may be set or reset by an Owner authorized command (TPM\_SetOwnerInstall 8.12.1). It may be set by a local command (TPM\_PhysicalDisable 8.13.2). It may be reset by a physical action (TPM\_PhysicalEnable 8.13.3).

These methods permit the Owner to disable the TPM when necessary (provided the TPM is accepting authorized commands from the Owner); permit a User or a Owner to disable a TPM via local access to the platform; and permit a User or Owner to activate a TPM by the use of physical access to the platform (which may or may not be trivial).

The TPM is disabled by a command that has originated locally. It may be that this “local” requirement restricts the operation of this command to times before an OS is running. The TPM is also disabled by an Owner authorized command. It may be that this “authorization” requirement restricts this command to times after the OS is running.

The TPM can be enabled by a physical event at the platform (whether or not the TPM has an Owner, and whether or not the OS is running). The TPM can also be enabled by an Owner authorized command. It may be that this “authorization” requirement restricts this command to times after the OS is running.

(BIT 2) This may be set or reset by a local command (TPM\_SetOwnerInstall 8.12.1).

This method permits a User or Owner to enable or disable the process of taking ownership, via local access to the platform. It may be that this “local” requirement restricts the operation of this command to times before an OS is running.

(BIT 3) This may be set or reset by a local command (TPM\_PhysicalSetDeactivated 8.14.1).

This method permits a User or an Owner to set the default active/deactive state of a TPM via local access to the platform. It may be that this “local” requirement restricts the operation of these commands to times before an OS is running.

(BIT 4) This may be set by a local command (TPM\_SetTempDeactivated 8.14.2). Any alteration lasts until the next boot cycle, when this bit is initialized to the state of BIT3.

This method permits a User or the Owner to temporarily deactivate the TPM. An unauthorized command causes the TPM to enter an inactive state. The TPM remains in that state until the platform is rebooted.

The default states of the persistent bits (BIT 1, 2, 3) in a virgin platform are the choice of the supplier. In a platform where “physical access” involves opening the platform, a supplier may wish to set DISABLE-TPM=FALSE, for example. In a platform where the supplier knows that the customer will use the Subsystem, a supplier may wish to set DISABLED\_OWNER\_INSTALL=FALSE and DEACTIVATED\_TPM=FALSE, for example. In a platform where the supplier is uncertain whether the customer will use the Subsystem, a supplier may wish to set DISABLED\_OWNER\_INSTALL=TRUE and DEACTIVATED\_TPM=TRUE, for example.

Both a disabled TPM and an inactive TPM never prevent the “extend” capability from operating. This is necessary in order to ensure that the records of sequences of integrity metrics in a TPM are always up-to-date.

***End of informative comment.***

## 3. Protection

### 3.1 Introduction

***Start of informative comment:***

The Protection Profile in the Conformance part of the specification defines the threats that are resisted by a platform. This section, "Protection," describes the properties of selected capabilities and selected data locations within a platform that has a Protection Profile and has not been modified by physical means.

This section introduces the concept of protected capabilities and the concept of shielded locations for data. Every definition of a TCPA capability states whether it is a protected capability. Data definitions state whether the data must be held in shielded locations.

- A protected capability is one whose correct operation is necessary in order for the operation of the Subsystem to be trusted.
- A shielded location is an area where data is protected against interference and prying, independent of its form.

This specification uses the concept of protected capabilities so as to distinguish those Subsystem capabilities that must be trustworthy. Trust in the Subsystem depends critically on the protected capabilities. Subsystem capabilities that are not protected capabilities must (of course) work properly if the Subsystem is to function properly.

This specification uses the concept of shielded locations, rather than the concept of "shielded data." While the concept of shielded data is intuitive, it is extraordinarily difficult to define because of the imprecise meaning of the word "data." For example, consider data that is produced in a safe location and then moved into ordinary storage. It is the same data in both locations, but in one it is shielded data and in the other it is not. Also, data may not always exist in the same form. For example, it may exist as vulnerable plaintext, but also may sometimes be transformed into a logically protected form. This data continues to exist, but doesn't always need to be shielded data - the vulnerable form needs to be shielded data, but the logically protected form does not. If a specific form of data requires protection against interference or prying, it is therefore necessary to say "if the data-D exists, it must exist only in a shielded location." A more concise expression is "the data-D must be extant only in a shielded location."

Hence if trust in the Subsystem depends critically on access to certain data, that data should be extant only in a shielded location and accessible only to protected capabilities. When not in use, such data could be erased after conversion (using a protected capability) into another data structure. Unless the other data structure was defined as one that must be held in a shielded location, it need not be held in a shielded location.

***End of informative comment.***

### 3.2 Threat

***Start of informative comment:***

This section, "Threat," defines the scope of the threats that must be considered when considering whether a platform facilitates subversion of capabilities and data in a platform.

The design and implementation of a platform determines the extent to which the platform facilitates subversion of capabilities and data within that platform. It is necessary to define the attacks that must be resisted by TCPA-shielded locations and TCPA-protected capabilities in that platform.

The TPM Protection Profile defines all attacks that are resisted by the TPM. These attacks must be considered when determining whether the integrity of TCPA-protected capabilities and data in TCPA-shielded locations can be damaged. These attacks must be considered when determining whether there is a backdoor method of obtaining access to TCPA-protected capabilities and data in TCPA-shielded locations. These attacks must be considered when determining whether TCPA-protected capabilities have undesirable side effects.

***End of informative comment.***

For the purposes of the “Protection” section of the specification: the threats that **MUST** be considered when determining whether the platform facilitates subversion of TCPA-protected capabilities or data in TCPA-shielded locations **SHALL** include the methods inherent in physical attacks that should fail if the platform complies with its protection profile, and **SHALL** include all methods that require execution of instructions in a computing engine in the platform.

### 3.3 Integrity

***Start of informative comment:***

A TCPA-protected capability must be used to modify TCPA-protected capabilities or data in TCPA-shielded locations. Other methods must not be allowed to modify TCPA-protected capabilities or data in TCPA-shielded locations. Otherwise, the integrity of TCPA-protected capabilities and data in TCPA-shielded locations is unknown.

***End of informative comment.***

A platform **SHALL NOT** facilitate the alteration of TCPA-protected capabilities or data in TCPA-shielded locations, except by TCPA-protected capabilities.

### 3.4 Privileged Access

***Start of informative comment:***

Only TCPA-protected capabilities are allowed to use the data in TCPA-shielded locations. Otherwise, a rogue can pretend to be a TCPA entity.

***End of informative comment.***

A platform **SHALL NOT** facilitate the disclosure or the exposure of data in TCPA-shielded locations, except to TCPA-protected capabilities.

### 3.5 Side effects

***Start of informative comment:***

An implementation of a TCPA-protected capability must not disclose the contents of TCPA-shielded locations. The only exceptions are when such disclosure is inherent in the definition of the capability or in the methods used by the capability. For example, a capability might be designed specifically to reveal hidden data or might use cryptography and hence always be vulnerable to cryptanalysis. In such cases, some disclosure or risk of disclosure is inherent and cannot be avoided. Other forms of disclosure (by side effects, for example) must always be avoided.

***End of informative comment.***

The implementation of a TCPA-protected capability in a platform **SHALL NOT** facilitate the disclosure or the exposure of data in TCPA-shielded locations except by means unavoidably inherent in the TCPA definition.

## 4. Structures and Defines

***Start of informative comment:***

The following structures and formats describe the interoperable areas of the specification. There is no requirement that internal storage or memory representations of data must follow these structures. These requirements are in place only during the movement of data from a TPM to some other entity.

***End of informative comment.***

### 4.1.1 Endness of Structures

Each structure MUST use big endian bit ordering, which follows the Internet standard and requires that the low-order bit appear to the far right of a word, buffer, wire format, or other area and the high-order bit appear to the far left.

### 4.1.2 Byte Packing

All structures MUST be packed on a byte boundary.



## 4.2 Defines

### *Start of informative comment:*

The defines are found in tcpa\_defines.h.

### *End of informative comment.*

### 4.2.1 Basic data types

#### Parameters

Typedef	Name	Description
unsigned char	BYTE	Basic byte used to transmit all character fields.
unsigned char	BOOL	TRUE/FALSE field. TRUE = 0x01, FALSE = 0x00
unsigned short	UINT16	16 bit field. The definition in different architectures may need to specify 16 bits instead of the short definition
unsigned long	UINT32	32 bit field. The definition in different architectures may need to specify 32 bits instead of the long definition
	TRUE	0x01
	FALSE	0x00

### 4.2.2 Helper redefinitions

The following definitions are to make the IDL definitions more explicit and easier to read.

#### Parameters

Typedef	Name	Description
UINT32	TCPA_PCRINDEX	Index to a PCR register
UINT32	TCPA_DIRINDEX	Index to a DIR register
UINT32	TCPA_KEYHANDLE	Handle to a loaded key
UINT32	TCPA_AUTHHANDLE	Handle to an authorization session
UINT32	TSS_HASHHANDLE	Handle to a hash session
UINT32	TSS_HMACHANDLE	Handle to a HMAC session
UINT32	TCPA_ENCHANDLE	Handle to a encryption/decryption session
UINT32	TCPA_EVENTTYPE	Type of PCR event. See 4.15
UINT32	TCPA_COMMAND_CODE	The command ordinal. See 4.29
UINT32	TCPA_KEY_SLOT	The slot where a key is held. 0 Based
UINT16	TCPA_PROTOCOL_ID	The protocol in use. See 4.28
HRESULT	TCPA_RESULT	The return code from a function
BYTE	TCPA_AUTH_DATA_USAGE	When is authorization required for an entity

### 4.2.3 Vendor specific

For all items that can specify an individual algorithm, protocol or item the specification allows for vendor specific selections. The mechanism to specify a vendor specific mechanism is to set the high bit of the identifier on.

The following defines allow for the quick specification of a vendor specific item.

#### Parameters

Name	Value
TCPA_Vendor_Specific32	0x80000000
TCPA_Vendor_Specific16	0x8000
TCPA_Vendor_Specific8	0x80

### 4.3 Return codes

**Start of informative comment:**

All functions return a standard set of return values. These are the TCPA specific return codes (the values of TCPA\_RESULT).

HRESULT is a basic type of IDL RPC function calls. Basing the return codes on these values allows for the inclusion of the return code in the HMAC calculation when the TPM responds to a function.

**End of informative comment.**

**Parameters**

Name	Value	Description
TCPA_BASE	0x0	The start of TCPA return codes
TCPA_SUCCESS	TCPA_BASE	Successful completion of the operation
TCPA_AUTHFAIL	TCPA_BASE + 1	Authentication failed
TCPA_BADINDEX	TCPA_BASE + 2	The index to a PCR, DIR or other register is incorrect
TCPA_BAD_PARAMETER	TCPA_BASE + 3	One or more parameter is bad
TCPA_BUFSIZE	TCPA_BASE + 4	The size specified in MaxSize is not large enough to hold the data structure. If this error is returned, *Size is still set to the buffer size required.
TCPA_CLEAR_DISABLED	TCPA_BASE + 5	The clear disable flag is set and all clear operations now require physical access
TCPA_DEACTIVATED	TCPA_BASE + 6	The TPM is deactivated
TCPA_DISABLED	TCPA_BASE + 7	The TPM is disabled
TCPA_DISABLED_CMD	TCPA_BASE + 8	The target command has been disabled
TCPA_FAIL	TCPA_BASE + 9	The operation failed
TCPA_INACTIVE	TCPA_BASE + 10	The TPM is inactive
TCPA_INSTALL_DISABLED	TCPA_BASE + 11	The ability to install an owner is disabled
TCPA_INVALID_HANDLE	TCPA_BASE + 12	The handle presented was invalid
TCPA_KEYNOTFOUND	TCPA_BASE + 13	The target key was not found
TCPA_KEYNOTLOADED	TCPA_BASE + 14	No backup key is loaded
TCPA_MIGRATEFAIL	TCPA_BASE + 15	Migration authorization failed
TCPA_NO_PCR_INFO	TCPA_BASE + 16	A list of PCR values was not supplied
TCPA_NOSPACE	TCPA_BASE + 17	No room to load key.
TCPA_NOSRK	TCPA_BASE + 18	There is no SRK set
TCPA_NOTSEALED_BLOB	TCPA_BASE + 19	An encrypted blob is invalid or was not created by this TPM
TCPA_OWNER_SET	TCPA_BASE + 20	There is already an Owner
TCPA_RESOURCES	TCPA_BASE + 21	The event log is full

TCPA_SHORTRANDOM	TCPA_BASE + 22	A random string was too short
TCPA_SIZE	TCPA_BASE + 23	The TPM does not have the space to perform the operation.
TCPA_WRONGPCRVAL	TCPA_BASE + 25	The named PCR value does not match the current PCR value.
TCPA_BUSY	TCPA_BASE +26	The TPM is too busy to respond to the command

## 4.4 IDL

***Start of informative comment:***

To facilitate the definition of the messages blocks that can be properly authenticated the command definitions use IDL. The IDL defines should only use IN, OUT and size\_is.

The addition of the AUTH parameter is merely an indication in the specification as to which parameter should be included in the authorization calculation. The actual AUTH define is merely to blank.

***End of informative comment.*****Parameters**

Define	Name	Description
	in	Parameter is provided to function
	out	Parameter is returned from function
	in out	Parameter is both an input and output parameter
	AUTH	Blank marker to indicate the inclusion of the parameter in the authorization calculation
	Size_is	The size of a variable field

## 4.5 TCPA\_VERSION

For each structure in use externally by this specification, the following structure **MUST** be included.

### IDL Definition

```
typedef struct tdTCPA_VERSION
{
    BYTE Major;
    BYTE Minor;
    BYTE RevMajor;
    BYTE RevMinor;
} TCPA_VERSION;
```

### Parameters

Type	Name	Description
BYTE	major	This SHALL be the major version indicator. For version 1 this MUST be 0x01
BYTE	minor	This SHALL be the minor version indicator. For version 1 this MUST be 0x00
BYTE	RevMajor	This SHALL be the value of the TCPA_PERSISTENT_FLAGS.revMajor
BYTE	RevMinor	This SHALL be the value of the TCPA_PERSISTENT_FLAGS.revMinor

### Descriptions

The version points to the version of the specification that defines the structure.

## 4.6 TCPA\_DIGEST

### ***Start of informative comment:***

The digest value reports the result of a hash operation. In Version 1.0 of this specification the hash algorithm is SHA-1 with a resulting hash result being 160 bits.

### ***End of informative comment.***

### **Definition**

```
typedef struct tdTCPA_DIGEST{
    BYTE digest[20];
} TCPA_DIGEST;
```

### **Parameters**

Type	Name	Description
BYTE	digest	This SHALL be the actual digest information

### **Description**

For the SHA-1 hash the digestSize parameter MUST be 20. For all TCPA hash operations, the required algorithm is SHA-1.

For hash algorithms other than SHA-1 the digestSize parameter MUST indicate the block size of the algorithm and MUST be 20 or greater.

### **Redefinitions**

Typedef	Name	Description
TCPA_DIGEST	TCPA_PCRVALUE	The value inside of the PCR
TCPA_DIGEST	TCPA_BACKUP_AUTH	This SHALL be the digest of the concatenation TPM Owners authorization data and the public migration key
TCPA_DIGEST	TCPA_COMPOSITE_HASH	This SHALL be the hash of a list of PCR indexes and PCR values that a key or data is bound to (See 10.4.5 for details)
TCPA_DIGEST	TCPA_DIRVALUE	This SHALL be the value of a DIR register

## 4.7 TCPA\_NONCE

***Start of informative comment:***

A nonce is a random value that provides protection from replay and other attacks. Many of the commands and protocols in the specification require a nonce. This structure provides a consistent view of what a nonce is.

***End of informative comment.*****Definition**

```
typedef struct tdTCPA_NONCE{  
    BYTE nonce[20];  
} TCPA_NONCE;
```

**Parameters**

Type	Name	Description
BYTE	nonce	This SHALL be the 20 bytes of random data. When created by the TPM the value MUST be the next 20 bytes from the RNG.



## 4.8 TCPA\_AUTHDATA

### *Start of informative comment:*

The authorization data is the information that is saved or passed to provide proof of ownership of an entity.

For version 1.0 this area is always 20 bytes.

### *End of informative comment.*

### Definition

```
typedef struct tdTCPA_AUTHDATA{
    BYTE data[20];
} TCPA_AUTHDATA;
```

### Parameters

Type	Name	Description
BYTE	data	The data SHALL be the 20 bytes of information. The Owner can select any value for the data.

### Descriptions

When sending authorization data to the TPM the TPM does not validate the decryption of the data. It is the responsibility of the entity owner to validate that the authorization data was properly received by the TPM. This could be done by immediately attempting to open an authorization session.

### Redefinitions

Typedef	Name	Description
TCPA_AUTHDATA	TCPA_SECRET	A secret value used in the authorization process, this value is not encrypted
TCPA_AUTHDATA	TCPA_ENCAUTH	Encrypted auth data. The encryption mechanism is function dependent

## 4.9 TCPA\_PAYLOAD\_TYPE

### *Start of informative comment:*

To specify the type of payload in the TCPA\_STORE\_ASYM structure.

### *End of informative comment.*

### Definition

```
typedef unsigned char TCPA_PAYLOAD_TYPE;
```

### TCPA\_ENTITY\_TYPE Values

Value	Event Name	Comments
'A'	TCPA_PT_ASYM	The entity is an asymmetric key
'D'	TCPA_PT_DATA	The entity is data
'M'	TCPA_PT_MIGRATE	The entity is a migration blob

'T'	TCPA_PT_MAINT	The entity is a maintenance blob
-----	---------------	----------------------------------

## 4.10 TCPA\_INTERNAL\_HDR

### ***Start of informative comment:***

This structure applies header values to the encoded blob before the encryption process.

### ***End of informative comment.***

### **Definition**

```
typedef struct tdTCPA_INTERNAL_HDR {           // pos    len    total
    BYTE data;                                //    0      1      1
} TCPA_INTERNAL_HDR;
```

### **Parameters**

Type	Name	Description
BYTE	data	This SHALL be a magic number to ensure that RSA encryption will always succeed. The value MUST be 0x00

## 4.11 TCPA\_PUBKEY

***Start of informative comment:***

The TCPA\_PUBKEY structure contains the public portion of an asymmetric key pair. The algorithm identification comes from the TCPA\_KEY structure.

***End of informative comment.***

**Definition**

```
typedef struct tdTCPA_PUBKEY{
    UINT32 publen;
    [size_is(publen)] BYTE* pubkey;
} TCPA_PUBKEY;
```

**Parameters**

Type	Name	Description
UINT32	Publen	This SHALL be the length of the pubkey field
BYTE*	pubKey	This SHALL be the algorithm specific information

**Descriptions**

The algorithm specific information provides the public key for a specific algorithm.

The TPM MUST support TCPA\_RSA\_PUBKEY as the structure in parameter pubKey.

### 4.11.1 TCPA\_RSA\_PUBKEY

This structure is the pubkey parameter of TCPA\_PUBKEY when the algorithmID parameter of TCPA\_KEY is TCPA\_ALG\_RSA. It provides the algorithm specific information for an RSA public key.

**Definition**

```
typedef struct tdTCPA_RSA_PUBKEY{
    UINT32 modulusSize;
    [size_is(modulusSize)] BYTE* modulus;
} TCPA_RSA_PUBKEY;
```

**Parameters**

Type	Name	Description
UINT32	modulusSize	This SHALL be the size of the modulus
BYTE*	modulus	This SHALL be the modulus of the RSA public key

## 4.12 TCPA\_PRIVKEY

***Start of informative comment:***

The TCPA\_PRIVKEY contains the private portion of an asymmetric key pair. Most of the information is encrypted to provide security.

***End of informative comment.*****Definition**

```
typedef struct tdTCPA_PRIVKEY{
    UINT32 Privlen;
    [size_is(Privlen)] BYTE* Privkey;
} TCPA_PRIVKEY;
```

**Parameters**

Type	Name	Description
UINT32	Privlen	This SHALL be the length of the priv field
BYTE*	PrivKey	This SHALL be an encrypted TCPA_STORE_ASYMKEY

**Descriptions**

The TPM MUST store in the Privkey parameter only an encrypted TCPA\_STORE\_ASYMKEY structure.

## 4.13 TCPA\_KEY

### **Start of informative comment:**

The TCPA\_KEY structure provides a mechanism to transport the entire asymmetric key pair. The private portion of the key always is encrypted.

### **EndStart of informative comment.:**

### **Definition**

```
typedef struct tdTCPA_KEY{
    TCPA_VERSION ver;
    UINT32 algorithmID;
    UINT32 parmSize;
    [size_is(parmSize)] BYTE* parms;
    TCPA_PUBKEY pubKey;
    TCPA_PRIVKEY privKey;
} TCPA_KEY;
```

### **Parameters**

Type	Name	Description
TCPA_VERSION	ver	Version number defined in section 4.5.
UINT32	algorithmID	This SHALL be the type of algorithm in use
UINT32	parmSize	This SHALL be the length of the parms field.
BYTE*	parms	This SHALL be the algorithm specific parameters.
TCPA_PUBKEY	pubKey	This SHALL be the public portion of the key
TCPA_PRIVKEY	privKey	This SHALL be the private portion of the key

### **Descriptions**

The algorithm ID comes from the TCPA\_DEFINES.H file.

The TPM MUST support algorithm TCPA\_ALG\_RSA.

### **algorithmID equals TCPA\_ALG\_RSA**

The parms parameter MUST contain a pointer to a UNIT16 that contains the key lengthMUST contain the key size for the key pair.

## 4.14 TCPA\_PCR\_REGISTER

**Start of informative comment:**

TCPA\_PCR\_REGISTER is a structure used to return PCR contents and flags.

**End of informative comment.****IDL Definition**

```
typedef struct tdTCPA_PCR_REGISTER {  
    TCPA_PCRINDEX      Index  
    UINT32              Flags;  
    TCPA_PCRVALUE PCR;  
} TCPA_PCR_REGISTER;
```

**Parameters**

Type	Name	Description
TCPA_PCRINDEX	Index	Index of the PCR that is being read
UINT32	Flags	Flags register. Currently no flags are defined, so this parameter will always be set to zero.
TCPA_PCRVALUE	PCR	Set to current contents of the PCR

Note that the PCR index is explicit in this structure. TCPA-protected capabilities will set this index when returning this structure.

## 4.15 TCPA\_PCR\_EVENT

### **Start of informative comment:**

Individual events are stored in the T CPA\_PCR\_EVENT variably sized data structure.

### **End of informative comment.**

### **Definition**

```
typedef struct tdTCPA_PCR_EVENT {
    UINT32      Length;
    T CPA_PCRINDEX PCRIndex;
    [size_is(Length)] BYTE*      Event;
    T CPA_EVENTTYPE EventType;
    T CPA_PCRVALUE PcrValue
} T CPA_PCR_EVENT;
```

Where the structure members are as follows:

Type	Name	Description
UINT32	Length	Length of the event parameter
UINT32	PCRIndex	Index of the PCR to which this event belongs
BYTE*	Event	Variable-sized BYTE array
T CPA_EVENTTYPE	EventType	The type of event
T CPA_PCRVALUE	PcrValue	The value EXTENDED into the PCR

TCPA defines the following event/supporting information types:

### **EventType Values**

Value	Event Name	Comments
0	EV_CODE_CERT	The TPM_Extend event is in response to loading a firmware or software component for which a VE certificate was found. *Event points to the VE certificate that was shipped with the platform firmware or software (or discovered by other means). Size indicates the length of this structure. ExtendValue is the digest of the firmware, software or other code loaded.
1	EV_CODE_NOCERT	The event was in response to loading a firmware or other software component, but no VE certificate was found. The size is 0 and *Event is unused. However, ExtendValue is the digest of the firmware discovered. Absence of a VE certificate does not indicate lack of trust; it merely indicates that a VE certificate was not available at this point in boot. Upper-level software may be able to obtain such certificates.
2	EV_XML_CONFIG	The event describes the platform configuration. The supporting information is a platform or firmware-defined XML data structure that indicates security-relevant hardware configuration information. The event logged to TPM_Extend is the SHA-1 digest of the XML data structure, and the firmware guarantees that the configuration stated in the data structure is in effect when the firmware relinquishes control to the next module in boot. Size is the size in bytes of the XML data structure, and *Event points to the data structure itself. The information may include size of physical memory, number of processors, chipset configuration, buses discovered and processor/bus frequencies.

		Firmware vendors are free to define the XML reporting structure and select those parameters that are important for their platforms.
3	EV_NO_ACTION	The action was not performed. The corresponding DIGEST structure MUST be 0x1 (a single binary digit in the LSB of the DIGEST structure), and this value MUST also be logged to the TPM using the corresponding TPM_Extend operation. A supporting data structure may be supplied containing information that describes why the event did not occur. If such supporting information is supplied, it should be well-formed XML. However, this supporting information is not required.
4	EV_SEPARATOR	A list of actions was complete. This event must be used if more than one event can be logged to the TPM and upper-level software needs to be informed that logging was completed.
5 – $2^{16}-1$	Reserved	TCPA-reserved event types
$2^{16}$ – $2^{32}-1$	User-definable	Undefined and free for general-purpose use

Additional event types may be defined for TCPA usage in specific computing platforms (for example, the PC).



## 4.16 TCPA\_AUDIT\_EVENT structure

***Start of informative comment:***

This structure reports the contents of the audit log. The entries in the log, if hashed together should equal the current hash value held by the TPM. Mismatches indicate attacks on the system or failures to properly audit events.

The 1.0 version has the minimal information necessary to recreate the history of audited operations.

Future versions may add additional information.

***End of informative comment.*****IDL Definition**

```
typedef struct tdTCPA_AUDIT_EVENT{
    TCPA_COMMAND_CODE ordinal;
    TCPA_RESULT returncode;
} TCPA_AUDIT_EVENT;
```

**Parameters**

Type	Name	Description
TCPA_COMMAND_CODE	ordinal	Ordinal of the command
TCPA_RESULT	returncode	Return code for the command

## 4.17 Storage Structures

### 4.17.1 TCPA\_SEALED\_DATA

***Start of informative comment:***

The definition of this structure is necessary to ensure the enforcement of security properties.

This structure is in use by the TPM\_Seal and TPM\_Unseal commands to identify the PCR index and values that must be present to properly unseal the data.

***End of informative comment.***

**Definition**

```
typedef struct tdTCPA_SEALED_DATA {
    TCPA_VERSION ver;
    TCPA_MAGIC1 MagicNumber;
    BOOL IsSealedToPCR;
    UINT32 dataSize;
    TCPA_COMPOSITE_HASH digestAtCreation;
    TCPA_COMPOSITE_HASH digestAtUnseal;
    TCPA_SECRET authData;
    TCPA_DIGEST tpmProof;
    BYTE* data;
} TCPA_SEALED_DATA;
```

**Parameters**

Type	Name	Description
TCPA_VERSION	ver	Version number defined in section 4.5.
TCPA_MAGIC1	magicNumber	The bytes 0x15, 0x13, to distinguish TCPA_SEALED_DATA blobs from other data.
BOOL	IsSealedToPCR	This SHALL be TRUE or FALSE. If set to FALSE, a TPM_Unseal command will not check digestAtUnseal against PCR values
UINT32	dataSize	This SHALL be the size of the data parameter
TCPA_COMPOSITE_HASH	digestAtCreation	This SHALL be the composite digest value of the values, at the time when the seal is performed, of the PCR registers to which parameter data is sealed.
TCPA_COMPOSITE_HASH	digestAtUnseal	This SHALL be the composite digest value of the PCR register values to which parameter data is sealed.
TCPA_SECRET	authData	This SHALL be the authorization data for this value
TCPA_DIGEST	tpmProof	This SHALL a copy of TPM_PERSISTENT_FLAGS.tmpProof
BYTE*	data	This SHALL be the data to be sealed

**Descriptions**

This entire structure is encrypted during the TPM\_Seal process. When the TPM\_Unseal decrypts this structure the TPM\_Unseal uses the information in the structure to validate the current configuration and release the decrypted data.

**Magic number**

```
typedef struct tdTCPA_MAGIC1{  
    BYTE num[2] = 0x15,0x13  
} TCPA_MAGIC1
```

### 4.17.2 TCPA\_PCR\_SELECTION

***Start of informative comment:***

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

***End of informative comment.***

**Definition**

```
typedef struct tdTCPA_PCR_SELECTION {
    TCPA_VERSION ver;
    BYTE pcrSelect[16];
} TCPA_PCR_SELECTION;
```

**Parameters**

Type	Name	Description
TCPA_VERSION	ver	Version number defined in section 4.5. This field is present so that if the number of available PCR registers changes this structure can accommodate the change.
BYTE	pcrSelect	This SHALL be a bit map that indicates if a PCR is active or not

**Description**

When the least-significant-bit of byte [N+1] of pcrSelect is butted against the most-significant-bit of byte [N] of pcrSelect for (15>=N>=0), the contiguous bit array so formed SHALL represent PCR indices in monotonically increasing order, starting from PCR index zero represented by bit 0 of byte 0 of pcrSelect.

The state of each bit in pcrSelect indicates whether a PCR register is selected or not. When the bit is 1 then the corresponding PCR is selected, if 0 the PCR is not selected.

This structure allows for the selection of up to 128 PCR registers.

### 4.17.3 TCPA\_PCR\_COMPOSITE

***Start of informative comment:***

The composite structure provides the index and value of the PCR register to be used when creating the value that SEALS an entity to the composite.

***End of informative comment.***

**Definition**

```
typedef struct tdTCPA_PCR_COMPOSITE {
    TCPA_PCR_SELECTION select;
    UINT32 valueSize;
    [size_is(valueSize)] BYTE* pcrValue;
} TCPA_PCR_COMPOSITE;
```

**Parameters**

Type	Name	Description
TCPA_PCR_SELECTION	select	This SHALL be the indication of which PCR values are active
BYTE	pcrValue	This SHALL be an array of TCPA_PCR_VALUE structures. The values come in the order specified by the select parameter and are concatenated into a single blob

### 4.17.4 TCPA\_KEY\_FLAGS

***Start of informative comment:***

This structure compacts the flag information in the TCPA\_STORE\_ASYMKEY structure.

***End of informative comment.***

**Definition**

```
typedef struct tdTCPA_KEY_FLAGS {
    unsigned IsWrappedToPCR : 1;
    unsigned Redirection      : 1;
    unsigned Migratable       : 1;
    unsigned Volatile         : 1;
    unsigned Migration        : 1;
    unsigned unused : 27;
} TCPA_KEY_FLAGS;
```

**Parameters**

Type	Name	Description
unsigned	IsWrappedToPCR	This SHALL indicate the use of PCRs. When FALSE the key SHALL NOT be associated with PCR values. When TRUE the key SHALL be associated with PCR value.
unsigned	Redirection	This SHALL indicate the use of redirected output. When FALSE the output SHALL use the normal output mechanism. When TRUE the output SHALL use a redirected output mechanism.
unsigned	Migratable	This SHALL indicate whether the key is migratable or not.

		When FALSE the key SHALL be non-migratable. When TRUE the key SHALL be migratable.
unsigned	Volatile	This SHALL indicate whether the key MUST be unloaded when the TPM is reset. When FALSE the TPM MUST unload the key upon reset. When TRUE the TPM MUST NOT unload the key upon reset.
unsigned	Migration	This SHALL indicate that this is a migration blob

### Description

For the purpose of this structure, the meaning of FALSE is the bit is off TRUE means the bit is on.

## 4.17.5 TCPA\_ASYM\_HASH

### *Start of informative comment:*

To allow for the size of TCPA\_STORE\_ASYMKEY to be under the modulus of a RSA 2048 bit key the hash value for the PCR composite digest must be a fixed length field.

This fixes the hash algorithm to SHA-1 and the size to 20 bytes.

### *End of informative comment.*

### Definition

```
typedef struct tdTCPA_ASYM_HASH{
    BYTE data[20];
} TCPA_ASYM_HASH;
```

### Parameters

Type	Name	Description
BYTE	data	The data SHALL be the 20 bytes of information

### Descriptions

The data SHALL be the result of a SHA-1 hash operation.

The data SHALL be the digest in the TCPA\_COMPOSITE\_HASH structure associated with this key.

#### 4.17.6 TCPA\_STORE\_ASYMKEY

**Start of informative comment:**

The TCPA\_STORE\_ASYMKEY structure provides the area to identity the private key factors of a asymmetric key.

The design of the structure is so that for RSA keys with a key size of 2048 can encrypt the structure in one operation.

Using typical RSA notation the structure would include P, and when loading the key include the unencrypted P\*Q which would be used to recover the Q value.

To accommodate the future use of multiple prime RSA keys the specification of additional prime factors is an optional capability.

The TPM\_KEY\_LEGACY key type is to allow for use in applications where both signing and encryptions operations occur with the same key. The use of this key type is deprecated.

This structure provides the basis of defining the protection of the private key. For the complete description of the entire encryption process, see 8.16.1.

**End of informative comment.**

Changes in this structure MUST be reflected in the TCPA\_MIGRATE\_ASYMKEY structure (section 4.17.7).

**Definition**

```
typedef struct tdTCPA_STORE_ASYMKEY {           // pos      len    total
    TCPA_ALGORITHM_ID AlgorithmID;              //    0        4      4
    TCPA_KEYUSAGE KeyUsage;                     //    4        2      6
    UINT32 typeTag;                             //    6        4     10
    UINT32 dataSize;                           //   10        4     14
    TCPA_SECRET auth;                          //   14       20     34
    TCPA_SECRET migration;                     //   34       20     54
    TCPA_ASYM_HASH pcrDigest;                  //   54       20     74
    TCPA_KEY_FLAGS keyFlags;                   //   74        4     78
    TCPA_AUTH_DATA_USAGE authDataUsage;        //   78        2     80
    BYTE data[];                               //   80      128    208
} TCPA_STORE_ASYMKEY;
```

**Parameters**

Type	Name	Description
TCPA_PAYLOAD_TYPE	ptTyp	This SHALL be the value from the payload type (key, data or migrate blob)
TCPA_ALGORITHM_ID	AlgorithmID	This SHALL be the algorithm identifier for the key in use.
TCPA_KEYUSAGE	KeyUsage	This SHALL be the TCPA key usage that determines the operations permitted with this key
UINT32	TypeTag	This SHALL be additional information regarding the algorithm.
UINT32	dataSize	This SHALL be the size of the data parameter.
TCPA_SECRET	Auth	This SHALL be the authorization data necessary to authorize the use of this value
TCPA_SECRET	Migration	This SHALL be the migration marker to prevent this item

		from migrating from one TPM to another. Implementation is left to TPM manufacturers.
TCPA_ASYM_HASH	PcrDigest	This SHALL be the digest of the PCR indices and PCR values to verify when loading the value. If IsWrappedToPCr is FALSE, this value is 20 bytes, each set to 0xFF.
TCPA_KEY_FLAGS	keyFlags	This SHALL be the indication of migration, redirection etc.
TCPA_AUTH_DATA_USAGE	AuthDataUsage	This SHALL indicate the authorization required upon each usage of the key
BYTE*	data	Actual private information dependent on key type. See descriptions

**TCPA\_KEYUSAGE values**

Name	Value	Description
TPM_KEY_SIGNING	0x0010	This SHALL indicate a signing key. The [private] key SHALL be used for signing operations, only. This means that it MUST be a leaf of the Protected Storage key hierarchy.
TPM_KEY_STORAGE	0x0011	This SHALL indicate a storage key. The key SHALL be used to wrap and unwrap other keys in the Protected Storage hierarchy, only.
TPM_KEY_IDENTITY	0x0012	This SHALL indicate an identity key. The key SHALL be used for operations that require a TPM identity, only.
TPM_KEY_LEGACY	0x0013	This SHALL indicate a key that can perform signing and decryption. The key MAY be used for both signing and decryption operations.
TPM_KEY_AUTHCHANGE	0x0014	This SHALL indicate an ephemeral key that is in use during the ChangeAuthAsym process, only.
TPM_KEY_DATA15	0x0015	This SHALL indicate a [private] key that may UNBIND a value in PKCS#1 1.5 version format, only
TPM_KEY_DATA20	0x0016	This SHALL indicate a [private] key that may UNBIND a value in PKCS#1 2.0 version format, only

**TCPA\_AUTH\_DATA\_USAGE values*****Start of informative comment:***

The method for providing universal access to an entity is to use a well known value for the authorization data.

***End of informative comment.***

Name	Value	Description
TPM_AUTH_ALWAYS	0x01	This SHALL indicate that on each usage of the key the authorization MUST be performed
		All other values are reserved for future use.

**Descriptions**

If AlgorithmID equals TCPA\_ALG\_RSA



The TypeTag parameter SHALL indicate the number of prime factors in use.

All migratable keys MUST be RSA keys with 2 prime factors.

#### **When TypeTag equals 2**

The TPM SHALL store and encrypt one of the prime factors in the TCPA\_STORE\_ASYMKEY structure. The data parameter MUST contain the prime factor for the key. Upon loading of the key the TPM calculates the other prime factor by dividing the modulus by this value.

#### **When TypeTag is greater than 2**

The TPM MAY support RSA keys with more than 2 prime factors.

### **Encryption**

#### ***Start of informative comment:***

The design of the TCPA\_STORE\_ASYMKEY structure holds a 2048 bit RSA key one encryption operation by a 2048 bit RSA key. This sets the maximum size of the TCPA\_STORE\_ASYMKEY structure as 208 bytes.

The encoding of the area by RSAES\_OAEP provides protections during migration.

#### ***End of informative comment.***

The design of the TCPA\_STORE\_ASYMKEY structure is such that for an RSA key of 2048 bits the key can be encrypted in one operation.

The TPM SHALL use the RSAES\_OAEP protocol from PKCS#1 version 2.0.

The following members of the ASYMKEY structure match the parameters in OAEP as follows:

#### **OAEP Parameters**

OAEP	TCPA_STORE_ASYMKEY	Description
SEED		The TPM SHALL provide the next 20 bytes from the TPM RNG
P	label	'TCPA' – a four (4) byte string
M	The TCPA_STORE_ASYMKEY area	The information to encrypt

#### 4.17.7 TCPA\_MIGRATE\_ASYMKEY

***Start of informative comment:***

The TCPA\_MIGRATE\_ASYMKEY structure provides the area to identity the private key factors of a asymmetric key while the key is migrating between TPM's.

The basis for this structure is the TCPA\_STORE\_ASYMKEY structure. The only difference between the two structures is the removal in this structure of the Migration field.

This structure provides the basis of defining the protection of the private key. For the complete description of the entire encryption process, see 7.2.11.

***End of informative comment.***

**Definition**

```
typedef struct tdTCPA_MIGRATE_ASYMKEY {           // pos      len    total
    UINT32 AlgorithmID;                           //    0         4      4
    TCPA_KEYUSAGE KeyUsage;                       //    4         2      6
    UINT32 typeTag;                               //    6         4     10
    UINT32 dataSize;                              //   10         4     14
    TCPA_SECRET auth;                             //   14        20     34
    TCPA_ASYM_HASH pcrDigest;                     //   34        20     54
    TCPA_KEY_FLAGS keyFlags;                      //   54         4     58
    TCPA_AUTH_DATA_USAGE authDataUsage;           //   58         2     60
    BYTE data[];                                  //   60       128    188
} TCPA_MIGRATE_ASYMKEY;
```

**Parameters**

Type	Name	Description
	All parameters	All fields MUST match the TCPA_STORE_ASYMKEY (section 4.17.6) fields with the exception of the Migration field which is absent.

#### 4.17.8 TCPA\_MAINTENANCE\_ASYMKEY

***Start of informative comment:***

***End of informative comment.***

##### Definition

```
typedef struct tdTCPA_MAINTENANCE_ASYMKEY {      // pos      len      total
    TCPA_NONCE tpmProof;                          //    0          20      20
    BYTE data[];                                  //   20         128     148
} TCPA_MAINTENANCE_ASYMKEY;
```

##### Parameters

Type	Name	Description
TCPA_NONCE	tpmProof	This SHALL be a copy of the TCPA_PERSISTENT_FLAGS.tpmProof
BYTE*	data	This SHALL be one of the primes of the SRK.

## 4.18 TCPA\_AUTH

### ***Start of informative comment:***

The authorization structure provides a standard method to represent the information that all functions requiring authorization need.

The handle is always just opaque data that the TPM uses to index to the session information.

The structure is input to the IDL file and hence all data has a set format.

It is the responsibility of the caller to properly fill out the authorization structure and properly generate the HMAC for the command. The HMAC always includes all fields in the structure except for the HMAC hash result.

### ***End of informative comment.***

### **IDL Definition**

```
typedef struct tdTCPA_AUTH{
    TCPA_AUTHHANDLE authHandle;
    TCPA_NONCE nonce;
    TCPA_DIGEST digest;
    BOOL continueFlag;
}TCPA_AUTH;
```

### **Parameters**

Type	Name	Description
TCPA_AUTHHANDLE	authHandle	The handle that the TPM uses to locate the session information that it maintains regarding this authorization session.
TCPA_NONCE	nonce	The nonce from the sender of the structure. For incoming packets, the caller sets this value. For outgoing packets, this value is set by the TPM.
TCPA_DIGEST	digest	The result of the HMAC calculation.
BOOL	continueFlag	Defines whether (TRUE) or not (FALSE) the TPM keeps the session open after execution of the command. May be set by TPM to FALSE in response to certain operations.

### **Description**

The TPM MUST read an incoming TCPA\_AUTH structure and generate the outgoing TCPA\_AUTH structure.

## 4.19 TCPA\_CERTIFY\_INFO Structure

### *Start of informative comment:*

When the TPM certifies a key, it must provide a signature with a TPM identity key on information that describes that key. This structure provides the mechanism to do so.

### *End of informative comment.*

### IDL Definition

```
typedef struct tdTCPA_CERTIFY_INFO{
    TCPA_VERSION Version;
    TCPA_KEY_FLAGS keyFlags;
    UINT32 typeTag;
    TCPA_AUTH_DATA_USAGE authDataUsage;
    TCPA_KEYUSAGE KeyUsage;
    TCPA_COMPOSITE_HASH DigestValue;
    TCPA_DIGEST pubkeyDigest;
    TCPA_NONCE Data;
    TCPA_PCR_SELECTION pcrList;
};
```

### Parameters

Type	Name	Description
TCPA_VERSION	Version	TCPA version structure; section 4.5.
TCPA_KEY_FLAGS	keyFlags	This SHALL be set to the same value as the corresponding parameter in the TCPA_STORE_ASYM structure that describes the private part of the public key that is being certified.
UINT32	typeTag	This SHALL be set to the same value as the corresponding parameter in the TCPA_STORE_ASYM structure that describes the private part of the public key that is being certified.
TCPA_AUTH_DATA_USAGE	authDataUsage	This SHALL be set to the same value as the corresponding parameter in the TCPA_STORE_ASYM structure that describes the private part of the public key that is being certified.
TCPA_KEYUSAGE	KeyUsage	This SHALL be set to the same value as the corresponding parameter in the TCPA_STORE_ASYM structure that describes the private part of the public key that is being certified.
TCPA_DIGEST	DigestValue	This SHALL be the result of the composite hash algorithm using pcrList for input. If the public key that is being certified is not bound to any PCRs, this SHALL be set to TCPA_CERTIFY_NOPCR.
TCPA_DIGEST	pubDigest	This SHALL be the hash of the public key being certified.
TCPA_NONCE	Data	This SHALL externally provided data.
TCPA_PCR_SELECTION	pcrList	This SHALL be the list of PCR indices that were used to compute the composite hash in DigestValue. This SHALL be an empty list (pcrList.pcrCount set to 0) when the public key that is being certified is not bound to any PCRs.

## 4.20 TCPA\_QUOTE\_INFO Structure

**Start of informative comment:**

This structure provides the mechanism for the TPM to quote the current values of a list of PCRs.

**End of informative comment.****IDL Definition**

```
typedef struct tdTCPA_QUOTE_INFO{
    TCPA_VERSION Version;
    BYTE fixed[4];
    TCPA_COMPOSITE_HASH DigestValue;
    TCPA_DIGEST ExternalData,
} TCPA_QUOTE_INFO;
```

**Parameters**

Type	Name	Description
TCPA_VERSION	Version	TCPA version structure; section 4.5.
BYTE	fixed	This SHALL always be the string 'QUOT'
TCPA_COMPOSITE_HASH	DigestValue	This SHALL be the result of the composite hash algorithm using the current values of the requested PCR indices.
TCPA_DIGEST	ExternalData	160 bits of externally supplied data

## 4.21 TCPA\_KEY\_INFO

### **Informative comment**

This structure provides the information regarding a key in response to a TPM\_GetCapability call.

### **End of informative comment.**

```
typedef struct tdTCPA_KEY_INFO{
    TCPA_VERSION Version;
    TCPA_KEY info;
    UINT32 typeTag;
    TCPA_KEYUSAGE KeyUsage;
    BOOL parentPCRStatus;
    TCPA_AUTH_DATA_USAGE authDataUsage;
    TCPA_PCR_SELECTION pcrList;
} TCPA_KEY_INFO;
```

### **Parameters**

Type	Name	Description
TCPA_VERSION	Version	TCPA version structure; section 4.5.
TCPA_KEY	info	The input structure contains all parameters except pubkey and privkey (which are NULL), to specify the size and type of the new key.
UINT32	typeTag	This SHALL be set to the same value as the corresponding parameter in the TCPA_STORE_ASYM structure that describes the private part of the public key that is being certified.
TCPA_KEYUSAGE	KeyUsage	This SHALL be set to the same value as the corresponding parameter in the TCPA_STORE_ASYM structure that describes the private part of the public key that is being certified.
BOOL	parentPCRStatus	This SHALL indicate if any parent key was wrapped to a PCR
TCPA_AUTH_DATA_USAGE	authDataUsage	This SHALL be set to the same value as the corresponding parameter in the TCPA_STORE_ASYM structure that describes the private part of the public key that is being certified.
TCPA_PCR_SELECTION	pcrList	This SHALL be the list of PCR indices that were used to compute the composite hash in DigestValue. This SHALL be an empty list (pcrList.pcrCount set to 0) when the public key that is being certified is not bound to any PCRs.

## 4.22 Flag Structures

### ***Informative comment***

The TPM maintains flags in volatile and non-volatile areas. These flags indicate the status of enabling, ownership and activation.

The setting of these flags follows the same rule, either the TPM Owner authorizes the setting of the flag or the command requires physical presence.

The physical presence assertion definition is a manufacturer option. There are many methods of making the assertion and manufacturers can select any number of options. The underlying theme is that no remote entity should be able to change the status of the TPM without either knowledge of the TPM Ownership authentication or physical presence next to the platform.

One method of providing the physical presence assertion is to have the TPM accept commands during a period when the operation of the platform is constrained. In a PC, the method might operate during the POST and require input from the user via the keyboard. The TPM would allow access to the command until execution of some critical point and the POST process informed the TPM that it should no longer accept the commands.

### ***End of informative comment.***



### 4.22.1 TCPA\_PERSISTENT\_FLAGS Structure

#### IDL Definition

```
typedef struct tdTCPA_PERSISTENT_FLAGS{
    BOOL disable;
    BOOL ownership;
    BOOL deactivated;
    BOOL readPubek;
    BOOL disableOwnerClear;
    BOOL AllowMaintenance;
    BYTE revMajor;
    BYTE revMinor;
    TCPA_NONCE tpmProof;
    TCPA_PUBKEY ManufacturerPub;
} TCPA_PERSISTENT_FLAGS;
```

#### Type

TPM shielded location

#### Parameters

Type	Name	Description
BOOL	disable	The state of the disable flag. See 8.13
BOOL	ownership	The ability to install an owner. See 8.12
BOOL	deactivated	The state of the active flag. See 8.14
BOOL	readPubek	The ability to read the PUBEK without owner authorization. See 9.2.2
BOOL	disableOwnerClear	If the owner authorized clear commands are active. See 8.9.6
BOOL	AllowMaintenance	Can the TPM Owner create a maintenance archive. See 7.2.14
BYTE	revMajor	This SHALL be the TPM major revision indicator. This SHALL only be set by the TPME.
BYTE	revMinor	This SHALL be the TPM minor revision indicator. This SHALL only be set by the TPME.
TCPA_NONCE	tpmProof	This SHALL be the random number that each TPM maintains to validate blobs in the SEAL and other processes.
TCPA_PUBKEY	ManufacturerPub	This SHALL be the manufacturers public key to use in the maintenance operations. If maintenance is not available in the TPM this field may be null.

#### Description

The data structure TCPA\_PERSISTENT\_FLAGS SHALL exist only in a TPM shielded-location and SHALL be non-volatile.

#### Disable flag

If disable has the value of TRUE, all commands except TPM\_GetCapability, TPM\_Extend and the TPM enabling capabilities SHALL return the value TPCA\_DISABLED.

**Ownership flag**

If ownership has the value of FALSE, then any attempt to install an owner fails with the error value TPCA\_INSTALL\_DISABLED.

**Deactivated flag**

This flag sets the state of TPCA\_VOLATILE\_FLAGS.deactivated upon initialization.

**readPubek**

If readPubek is TRUE then the TPM\_ReadPubek will return the PUBEK, if FALSE the command will return TPCA\_DISABLED\_CMD.

**DisableOwnerClear**

If disableOwnerClear is TRUE then the clear commands requiring owner authorization will return TPCA\_CLEAR\_DISABLED, if false the commands will execute.

### 4.22.2 TCPA\_VOLATILE\_FLAGS Structure

#### IDL Definition

```
typedef struct tdTCPA_VOLATILE_FLAGS{
    BOOL deactivated;
    BOOL disableForceClear;
} TCPA_VOLATILE_FLAGS;
```

#### Type

TPM shielded location

#### Parameters

Type	Name	Description
BOOL	Deactivated	The state of the active flag.
BOOL	DisableForceClear	The state of the force clear flag.

#### Action

The data structure TCPA\_VOLATILE\_FLAGS SHALL exist only in a TPM shielded-location and SHALL be volatile.

#### Deactivated flag

The TPM SHALL set the state of the TCPA\_VOLATILE\_FLAGS.deactivated to the state of TCPA\_PERSISTENT\_FLAGS.deactivated on each startup.

If deactivated is TRUE the following commands will execute with their normal protections

- TPM\_GetCapability
- TPM\_Extend
- TPM\_TakeOwnership
- TPM enabling and disabling
- TPM activation and deactivation

All other commands SHALL return TCPA\_DEACTIVATED.

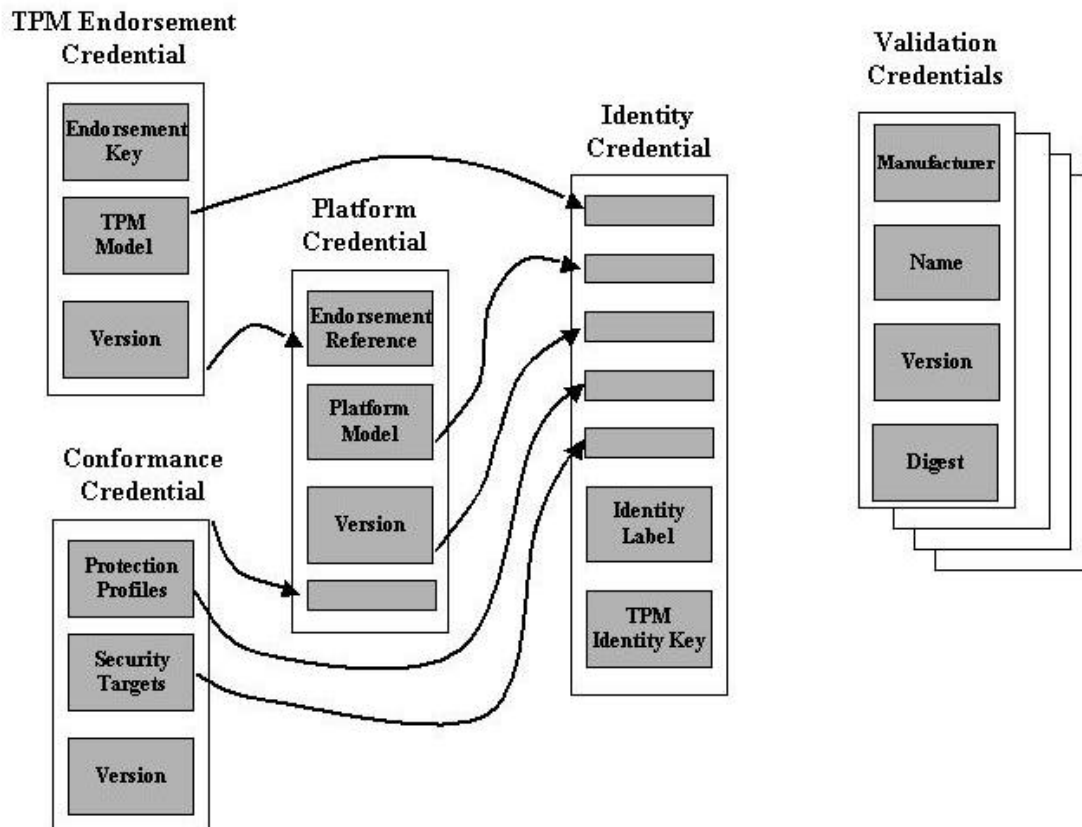
#### DisableForceClear

If disableForceClear is TRUE then the TPM\_ForceClear command returns TCPA\_CLEAR\_DISABLED, if FALSE then the command will execute.

## 4.23 Credentials

*Start of informative comment:*

The credentials in use for a TCPA system interlock. The following diagram shows the relationship between the credentials.



*End of informative comment.*

### 4.23.1 Evidence of Subsystem Endorsement

***Start of informative comment:***

The purpose of TPM\_ENDORSEMENT\_CREDENTIAL is to provide evidence that a TPM correctly implements the protected capabilities and shielded locations of the TCPA specification.

TPM\_ENDORSEMENT\_CREDENTIAL is an attestation that a genuine TCPA Trusted Platform Module created the PUBEK that is referenced in TPM\_ENDORSEMENT\_CREDENTIAL. TPM\_ENDORSEMENT\_CREDENTIAL contains information that a Privacy CA may use in judging whether the Privacy CA will attest to an identity of that TCPA Trusted Platform Module. TPM\_ENDORSEMENT\_CREDENTIAL contains information that the Privacy CA must use in attesting to an identity of that TCPA Trusted Platform Module.

TPM\_ENDORSEMENT\_CREDENTIAL is tagged with TCPA\_version so as to indicate the version of the capability that created the PUBEK at the time the key was generated. This may be useful in the event that capabilities are field-upgraded.

- PUBEK will be required by the Privacy CA when the Privacy CA attests to a TCPA Trusted Platform Module identity (TPM identity).
- “TCPA Trusted Platform Module Endorsement” identifies a data structure as TPM\_ENDORSEMENT\_CREDENTIAL and enables the TPME to sign the data with a key that is not exclusively reserved for signing TPM\_ENDORSEMENT\_CREDENTIAL.
- tpme\_reference is the means of referencing the TPME, may be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity, and is required by the Privacy CA when attesting to a TCPA TPM identity.
- tpm\_model is the means of referencing the type of implementation of protected capabilities and shielded locations. It may be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity and is required by the Privacy CA when attesting to a TCPA TPM identity.
- tpm\_distributed\_validation is a convenient immediate reference to the security properties of the implementation of protected capabilities and shielded locations. It may be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity and is required by the Privacy CA when attesting to a TCPA TPM identity.
- Access to the TPM\_ENDORSEMENT\_CREDENTIAL must be restricted to entities that have a “need to know.” This is for reasons of privacy.

***End of informative comment.***

**Description**

```
struct TPM_ENDORSEMENT_CREDENTIAL = {
    BYTE          label = "TCPA Trusted Platform Module Endorsement"
    TCPA_PUBKEY    public_endorsement_key
    REFERENCE      tpm_model
    REFERENCE      tpm_distributed_validation
    REFERENCE      tpme_reference
    TCPA_VERSION   TCPA_version
    SIGNATURE      signature_value}
```

This is an abstract definition, section 9.5.1 contains the concrete representation.

**Parameters**

Type	Name	Description
------	------	-------------

BYTE	Label	This SHALL be the ASCII characters "TCPA Trusted Platform Module Endorsement"
TCPA_PUBKEY	public_endorsement_key	This SHALL be the PUBKEY returned by a TPM_CreateEndorsementKeyPair command.
REFERENCE	tpm_model	This SHALL be a reference to the type of implementation of protected capabilities and shielded locations that created the PUBEK, plus a reference to the identity of the manufacturer of that implementation.
REFERENCE	tpm_distributed_validation	This SHALL be a reference to fields that indicate the security qualities of the implementation of protected capabilities and shielded locations that created the PUBEK.
REFERENCE	tpme_reference	This SHALL be an unambiguous indication of the identity of the (TPM) entity that attests that the implementation of protected capabilities and shielded locations conforms to the TCPA specification.
VERSION	TCPA_version	This SHALL be the version specified in section 4.5.
SIGNATURE	signature_value	This SHALL be the signature over all previous fields in TPM_ENDORSEMENT_CREDENTIAL, using the private key of the tpme-reference.

When an entity presents evidence to a Privacy CA that an implementation of protected capabilities and shielded locations conforms to the TCPA specification, that evidence SHALL include the data in the data structure TPM\_ENDORSEMENT\_CREDENTIAL.

A (TPME) entity SHALL NOT create the data structure TPM\_ENDORSEMENT\_CREDENTIAL unless the entity is satisfied that the PUBEK referenced in TPM\_ENDORSEMENT\_CREDENTIAL was returned in response to a TPM\_CreateEndorsementKeyPair command by an implementation of protected capabilities and shielded locations that meets the TCPA specification.

If the data structure TPM\_ENDORSEMENT\_CREDENTIAL is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.

### 4.23.2 Evidence of Platform Endorsement

***Start of informative comment:***

The purpose of platform\_credential is to provide evidence that a platform correctly incorporates an implementation of the protected capabilities and shielded locations of a TCPA Subsystem.

Platform\_credential is an attestation that a platform contains a genuine TCPA Subsystem. Platform\_credential contains information that a Privacy CA may use in judging whether the Privacy CA will attest to an identity of that TCPA Subsystem. Platform\_credential contains information that the Privacy CA must use in attesting to an identity of that TCPA Trusted Platform Subsystem.

Platform\_credential is tagged with TCPA\_version so as to indicate the version of the capability that created the PUBEK at the time that the key was generated. This may be useful in the event that capabilities are field-upgraded.

- TPM-reference is the means of referencing the specific implementation of protected capabilities and shielded locations that is incorporated into the platform. It will be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity
- The conformance-credential contains a set of conformance UIDs that unambiguously indicate the conformance to the TCPA specification of the TPM that is incorporated into the platform. These UIDs are the “tpm-protection-profile” and “tpm-security-target”. The conformance credential also contains a set of conformance UIDs that unambiguously indicate the conformance to the TCPA specification of the means by which the platform incorporates an implementation of the TPM, the implementation of the root-of-trust-for-measurement, and the means by which the platform incorporates an implementation of the root-of-trust-for-measurement. These UIDs are the “foundation-protection-profile” and “foundation-security-target”. All these UIDs will be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity.
- “TCPA Trusted Platform Endorsement” identifies a data structure as platform\_credential and enables the Platform Entity (PE) to sign the data with a key that is not exclusively reserved for signing platform\_credential.
- PE\_reference is the means of referencing the PE. It may be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity.
- platform\_model is the means of referencing the type of platform. The reference includes the implementation of TCPA foundations in the platform. The foundations include the root-of-trust-for-measurement that is incorporated into the platform, the method of incorporation of the RTM, and the method of incorporation of the TPM. It may be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity and is required by the Privacy CA when attesting to a TCPA TPM identity.
- platform\_distributed\_validation is a convenient immediate reference to the security properties of the platform. The reference includes the implementation of TCPA foundations in the platform. The foundations include the RTM that is incorporated into the platform, the method of incorporation of the RTM, and the method of incorporation of the TPM. It may be required by the Privacy CA when judging whether the Privacy CA will attest to a TCPA TPM identity and is required by the Privacy CA when attesting to a TCPA TPM identity.

Access to the platform\_credential must be restricted to entities that have a “need to know.” This is for reasons of privacy.

***End of informative comment.*****Description**

When an entity presents evidence to a Privacy CA that a platform conforms to the TCPA specification, that evidence SHALL include the data in the data structure platform\_credential.

An entity (PE) SHALL NOT create the data structure `platform_credential` unless the entity is satisfied that the platform conforms to the conformance credential referenced inside `platform_credential` and contains the TPM referenced inside `platform_credential`.

### Definition

```
struct PLATFORM_CREDENTIAL = {
    ASCII_STRING      "TCPA Trusted Platform Endorsement"
    REFERENCE         tpm-credential-reference
    REFERENCE         conformance-credential-reference
    REFERENCE         platform_model
    REFERENCE         platform_distributed_validation
    REFERENCE         pe-reference
    TCPA_VERSION       TCPA_version
    SIGNATURE          signature_value}
```

This is an abstract definition, section 9.5.2 contains the concrete representation.

### Parameters

Type	Name	Description
ASCII_STRING	"TCPA Trusted Platform Endorsement"	This SHALL be the ASCII string "TCPA Trusted Platform Endorsement"
REFERENCE	tpm-credential-reference	This SHALL be an unambiguous indication of the endorsement credential of the TPM incorporated into the platform.
REFERENCE	conformance-credential-reference	This SHALL be an unambiguous indication of the conformance UIDs that attest that the design of the platform conforms to the TCPA specification.
REFERENCE	platform_model	This SHALL be a reference to the type of the platform, including the TCPA foundations in the platform, plus a reference to the identity of the manufacturer of that platform.
REFERENCE	platform_distributed_validation	This SHALL be fields that indicate the general security qualities of the platform.
REFERENCE	pe-reference	This SHALL be an unambiguous indication of the identity of the (platform) entity that attests to the design and construction of the platform.
TCPA_VERSION	TCPA_version	This SHALL be the version specified in section 4.5.
SIGNATURE	signature_value	This SHALL be the signature over all previous fields in <code>platform_credential</code> , using the private key of the pe-reference.

If the data structure `platform_credential` is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.



### 4.23.3 Evidence of Platform Conformance

***Start of informative comment:***

The purpose of `conformance_credential` is to provide evidence that the design of the Subsystem in a platform correctly conforms to the TCPA specification, and that the design of the method of incorporation of the Subsystem in the platform correctly conforms to the TCPA specification.

`Conformance_credential` is an attestation that the overall design of a platform satisfies the TCPA specification. `Conformance_credential` contains information that a Privacy CA may use in judging whether the Privacy CA will attest to an identity of that TCPA Subsystem. `Conformance_credential` contains information that the Privacy CA must use in attesting to an identity of that TCPA Trusted Platform Subsystem.

`Conformance_credential` is tagged with `TCPA_version` so as to indicate the version of the capability that created the PUBKEY at the time that the key was generated. This may be useful in the event that capabilities are field-upgraded.

`Conformance_credential` contains identifiers (UIDs) that indicate the protection profile and the security target of both the TPM and the RTM, and the methods by which they are incorporated into the platform.

***End of informative comment.***

#### Description

When an entity presents evidence to a Privacy CA that a platform conforms to the TCPA specification, that evidence SHALL include the data in the data structure `conformance_credential`.

A (conformance) entity SHALL NOT create the data structure `conformance_credential` unless the entity is satisfied that the design of both the Subsystem and its incorporation into the platform are accurately and unambiguously represented by the information in `conformance_credential`.

```
typedef struct CONFORMANCE_CREDENTIAL = {
    ASCII_STRING      "TCPA Conformance Credential"
    CONFORM_UID       tpm_pp
    CONFORM_UID       tpm_st
    CONFORM_UID       foundation_pp
    CONFORM_UID       foundation_st
    REFERENCE         ce_reference
    TCPA_VERSION       TCPA_version
    SIGNATURE         signature
}
```

This is an abstract definition; section 9.5 contains the concrete representation.

#### Parameters

Type	Name	Description
ASCII_STRING	"TCPA Conformance Credential"	This SHALL be the ASCII string "TCPA Conformance Credential"
CONFORM_UID	tpm_pp	This SHALL be the UID that unambiguously identifies the protection profile of the TPM
CONFORM_UID	tpm_st	This SHALL be the UID that unambiguously identifies the security target of the TPM
CONFORM_UID	foundation_pp	This SHALL be the UID that unambiguously identifies the protection profile of the TCPA foundations in the platform.
CONFORM_UID	foundation_st	This SHALL be the UID that unambiguously identifies the security target of the TCPA foundations in the platform.

		identifies the security target of the TCPA foundations in the platform.
REFERENCE	ce_reference	This SHALL be an unambiguous indication of the identity of the (Conformance) entity that attests to the overall design of the platform.
TCPA_VERSION	TCPA_version	This SHALL be the version specified in section 4.5.
SIGNATURE	signature_value	This SHALL be the signature over all previous fields in CONFORMANCE_CREDENTIAL, using the private key of the ce_reference.

#### 4.23.4 TCPA Validation Data

***Start of informative comment:***

The purpose of TCPA Validation Data is to state the values of integrity metrics that should be obtained when the component described by the validation data is working properly.

TCPA Validation Data identifies a data structure as `validation_data` and enables the PE to sign the data with a key that is not exclusively reserved for signing `validation_data`.

***End of informative comment.***

All components that influence the software environment in a platform SHOULD have corresponding validation data.

The representation of a component SHALL reflect the way that the component influences the software environment in a platform. All representations SHALL include a description of the manufacturer, the common name of the component, the version of the component, and a field that describes the security qualities of the component.

The representation of a component SHALL NOT in any way provide information that exposes the identity of a specific component.

The validation data of a component SHALL be `validation_data`

**IDL Description**

```
typedef struct VALIDATION_DATA ={
    ASCII_STRING      "TCPA Validation Data"
    ASCII_STRING      component_manufacturer,
    ASCII_STRING      component_name,
    ASCII_STRING      component_version,
    DIGEST            instruction_digest,
    REFERENCE          component_distributed_validation,
    REFERENCE          ve_reference,
    TCPA_VERSION       TCPA_version,
    SIGNATURE          validation_data_signature_value}
```

This is an abstract definition; section 9.5.4 contains the concrete representation.

**Parameters**

Type	Name	Description
ASCII_STRING	"TCPA Validation Data"	This SHALL be the ASCII string "TCPA Validation Data."
ASCII_STRING	<code>component_manufacturer</code>	This SHALL be an ASCII string stating the name of the manufacturer of the component.
ASCII_STRING	<code>component_name</code>	This SHALL be an ASCII string stating the common name of the component.
ASCII_STRING	<code>component_version</code>	This SHALL be an ASCII string stating the version of the component.
DIGEST	<code>instruction_digest</code>	This SHALL be a digest of any instructions in the component that are intended to execute on the main computing engine of the platform.
REFERENCE	<code>component_distributed_validation</code>	This SHALL be a convenient immediate reference to the security properties of the

		reference to the security properties of the component.
REFERENCE	ve_reference	This SHALL be an unambiguous indication of the identity of the (validation) entity that attests to the validation data.
TCPA_VERSION	TCPA_version	This SHALL be the version specified in section 4.5.
SIGNATURE	validation_data_signature_value	This SHALL be the result of signing all fields (except this field) in VALIDATION_DATA using the signature (private) key of VE_reference.

#### 4.23.5 Evidence of Trusted Platform Module Identity

##### ***Start of informative comment:***

The data in TPM\_IDENTITY\_CREDENTIAL is presented whenever an entity requires proof that an anonymous identity belongs to a genuine TCPA Subsystem.

TPM\_IDENTITY\_CREDENTIAL may be accompanied by other data, depending upon circumstances. When presented in response to an integrity challenge, it may be accompanied by conventional certificates and validation data, for example.

TPM\_IDENTITY\_CREDENTIAL is tagged with TCPA\_version so as to indicate the version of the capability that created the identity key at the time that the key was generated. This may be useful in the event that capabilities are field-upgraded.

The phrase “TCPA Trusted Platform Module identity” identifies a data structure as a Trusted Platform Module identity and enables the Privacy CA to sign the data with a key that is not exclusively reserved for signing TPM identities.

Access to the TPM\_IDENTITY\_CREDENTIAL must be restricted to entities that have a “need to know.” This is for reasons of privacy.

##### ***End of informative comment.***

#### **Description**

When an entity presents evidence that an identity belongs to a Subsystem, that evidence SHALL include the data in the data structure TPM\_IDENTITY\_CREDENTIAL.

```
struct TPM_IDENTITY_CREDENTIAL ={
    ASCII_STRING      "TCPA Trusted Platform Identity"
    UNICODE           identityLabel
    TCPA_PUBKEY       identityPubKey
    REFERENCE         tpm_model
    REFERENCE         tpm_distributed_validation
    CONFORM_UID       tpm_pp
    CONFORM_UID       tpm_st
    REFERENCE         platform_model
    REFERENCE         platform_distributed_validation
    CONFORM_UID       foundation_pp
    CONFORM_UID       foundation_st
    REFERENCE         p-ca_reference
    TCPA_VERSION      TCPA_version
    SIGNATURE         signature_value}
```

This is an abstract definition; section 9.5.5 contains the concrete representation.

**Parameters**

Type	Name	Description
ASCII_STRING	"TCPA Trusted Platform Module Identity"	This SHALL be the ASCII string "TCPA Trusted Platform Identity."
UNICODE	identityLabel	This SHALL be a textual string associated with the TPM identity.
TCPA_PUBKEY	identityPubKey	This SHALL be a public key associated with the TPM identity.
REFERENCE	tpm_model	This SHALL be a reference to the type of TPM in the platform, plus a reference to the identity of the manufacturer of TPM.
REFERENCE	tpm_distributed_validation	This SHALL be fields that indicate the security qualities of the TPM in the platform.
CONFORM_UID	tpm_pp	This SHALL be the UID that unambiguously identifies the protection profile of the TPM
CONFORM_UID	tpm_st	This SHALL be the UID that unambiguously identifies the security target of the TPM
REFERENCE	platform_model	This SHALL be a reference to the type of the platform, including the TCPA foundations in the platform, plus a reference to the identity of the manufacturer of that platform.
REFERENCE	platform_distributed_validation	This SHALL be fields that indicate the security qualities of the platform.
CONFORM_UID	foundation_pp	This SHALL be the UID that unambiguously identifies the protection profile of the TCPA foundations in the platform.
CONFORM_UID	foundation_st	This SHALL be the UID that unambiguously identifies the security target of the TCPA foundations in the platform.
REFERENCE	p-ca_reference	This SHALL be an unambiguous indication of the identity of the (Privacy CA) entity that attests to the TPM identity.
TCPA_VERSION	TCPA_version	This SHALL be the version specified in section 4.5.
SIGNATURE	signature_value	This SHALL be the signature over all previous fields in TPM_IDENTITY_CREDENTIAL, using the private key of the p-ca_reference.

If the data structure TPM\_IDENTITY\_CREDENTIAL is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.

## 4.24 TCPA\_ALGORITHM\_PARMS

### *Start of informative comment:*

This provides a standard mechanism to define the algorithm parameters

### *End of informative comment.*

#### Definition

```
typedef struct tdTCPA_ALGORITHM_PARMS {
    UINT32 algorithmID;
    UINT32 parmSize;
    [size_is(parmSize)] BYTE* parms;
} TCPA_ALGORITHM_PARMS;
```

#### Parameters

Type	Name	Description
UINT32	algorithmID	This SHALL be the algorithm in use
UINT32	parmSize	This SHALL be the size of the parms field in bytes
BYTE*	parms	This SHALL be the parameter information

#### Descriptions

Name	Value	Description	parm Contents
TCPA_ALG_RSA	0x00000001	The RSA algorithm.	UINT32 Size of modulus
TCPA_ALG_DES	0x00000002	The DES algorithm	IV value for CBC calculation
TCPA_ALG_3DES	0x00000003	The 3DES algorithm	IV value for CBC calculation
TCPA_ALG_AES	0x00000004	The AES algorithm	IV value for CBC calculation
TCPA_ALG_SHA	0x00000005	The SHA1 algorithm	ignored
TCPA_ALG_HMAC	0x00000006	The HMAC algorithm	ignored

#### **algorithmID equals TCPA\_ALG\_RSA**

The parms field contains a UINT32 that specifies the size of the RSA key in bits.

#### **algorithmID equals TCPA\_ALG\_DES, TCPA\_ALG\_3DES, TCPA\_ALG\_AES**

The parms field contains a TCPA\_SYMMETRIC\_KEY that is the IV value for a CBC calculation.

## 4.25 Identity Structures

### 4.25.1 TCPA\_IDENTITY\_CONTENTS

***Start of informative comment:***

The TPM\_MakeIdentity uses this structure and the signature of this structure goes to a privacy CA during the certification process.

***End of informative comment.***

**Definition**

```
typedef struct tdTCPA_IDENTITY_CONTENTS {
    TCPA_VERSION        ver
    UINT32               ordinal,
    UINT32               labelSize,
    TCPA_PUBKEY          caPubKey,
    TCPA_PUBKEY          identityPubKey,
    [size_is(labelSize)] BYTE* identityLabel;
} TCPA_IDENTITY_CONTENTS;
```

**Parameters**

Type	Name	Description
TCPA_VERSION	ver	This SHALL be the version specified in section 4.5.
UINT32	ordinal	This SHALL be the ordinal of the TPM_MakeIdentity command.
UINT32	labelSize	This SHALL be the size of the identityLabel field
TCPA_PUBKEY	CaPubKey	This SHALL be the caPubKey field in the calling TPM_MakeIdentity command.
TCPA_PUBKEY	identityPubKey	This SHALL be the public key structure of the identity key
BYTE*	identityLabel	This SHALL be the label for the identityPubKey

#### 4.25.2 TCPA\_SYMMETRIC\_KEY

**Start of informative comment:**

This structure describes a symmetric key.

**End of informative comment.****Definition**

```
typedef struct tdTCPA_SYMMETRIC_KEY {  
    UINT16 size;  
    [size_is(size)] BYTE* data;  
} TCPA_SYMMETRIC_KEY;
```

**Parameters**

Type	Name	Description
UINT16	size	This SHALL be the size of the data parameter in bytes
BYTE*	data	This SHALL be the symmetric key data



### 4.25.3 TCPA\_IDENTITY\_REQ

**Start of informative comment:**

This structure is sent by the TSS to the Privacy CA to create the identity credential.

**End of informative comment.**

**Definition**

```
typedef struct tdTCPA_IDENTITY_REQ {
    UINT32      asymSize;
    UINT32      symSize;
    UINT32      asymAlg;
    TCPA_ALGORITHM_PARMS algorithm;
    [size_is(asymSize)] BYTE* asymBlob;
    [size_is(symSize)] BYTE* symBlob;
} TCPA_IDENTITY_REQ;
```

**Parameters**

Type	Name	Description
UINT32	asymSize	This SHALL be the size of the asymmetric encrypted area created by TSS_CollatIdentityRequest
UINT32	symSize	This SHALL be the size of the symmetric encrypted area created by TSS_CollatIdentityRequest
TCPA_ALGORITHM_PARMS	algorithm	This SHALL be the parameters for the symmetric algorithm
BYTE*	asymBlob	This SHALL be the asymmetric encrypted area from TSS_CollatIdentityRequest
BYTE*	symBlob	This SHALL be the symmetric encrypted area from TSS_CollatIdentityRequest

**Actions**

For reasons of interoperability, the asymmetric algorithm SHOULD be RSA with a key length of 2048 bits and the symmetric algorithm 3DES in CBC mode.

The algParms area MUST contain the IV value for the CBC encryption. The IV MUST be a random nonce from the TPM RNG.

The use of AES in CBC mode as the symmetric algorithm is encouraged.

#### 4.25.4 TCPA\_SYM\_IDENTITY\_REQ

**Start of informative comment:**

This structure is used during the process 9.3.2 “Collating a Request for a Trusted Platform Module Identity”

**End of informative comment.**

**Definition**

```
typedef struct tdTCPA_SYM_IDENTITY_REQ {
    TCPA_VERSION ver;
    TCPA_NONCE random;
    UINT32 labelSize;
    UINT32 identitySize;
    UINT32 endorsementSize;
    UINT32 platformSize;
    UINT32 conformanceSize;
    TCPA_PUBKEY caPubKey;
    [size_is(labelSize)] BYTE* labelArea;
    [size_is(identitySize)] BYTE* identityBinding;
    [size_is(endorsementSize)] BYTE* endorsementCredential;
    [size_is(platformSize)] BYTE* platformCredential;
    [size_is(conformanceSize)] BYTE* conformanceCredential;
} TCPA_SYM_IDENTITY_REQ;
```

Type	Name	Description
TCPA_VERSION	ver	This SHALL be the version specified in section 4.5.
TCPA_NONCE	Random	This SHALL be a nonce that has been created by a TCPA-protected capability
UINT32	labelSize	This SHALL be the size of the label area
UINT32	IdentitySize	This SHALL be the size of the identity area
UINT32	endorsementSize	This SHALL be the size of the endorsement credential
UINT32	PlatformSize	This SHALL be the size of the platform credential
UINT32	conformanceSize	This SHALL be the size of the conformance credential
TCPA_PUBKEY	CaPubKey	This SHALL be public key of the CA which will provide the credential for the identity
BYTE*	LabelArea	This SHALL be the text label for the new identity
BYTE*	IdentityBinding	This SHALL be the signature value of TCPA_IDENTITY_CONTENTS structure from the TPM_MakeIdentity command
BYTE*	endorsementCredential	This SHALL be the TPM endorsement credential
BYTE*	platformCredential	This SHALL be the TPM platform credential
BYTE*	conformanceCredential	This SHALL be the TPM conformance credential

#### 4.25.5 TCPA\_ASYM\_IDENTITY\_REQ

**Start of informative comment:**

This structure contains the symmetric key to encrypt the identity request.

**End of informative comment.****Definition**

```
typedef struct tdTCPA_ASYM_IDENTITY_REQ {  
    TCPA_SYMMETRIC_KEY sessionKey;  
} TCPA_ASYM_IDENTITY_REQ;
```

**Parameters**

Type	Name	Description
TCPA_NONCE	SessionKey	This SHALL be the session key

#### 4.25.6 TCPA\_ASYM\_CA\_CONTENTS

**Start of informative comment:**

This structure contains the symmetric key to encrypt the identity credential.

**End of informative comment.****Definition**

```
typedef struct tdTCPA_ASYM_CA_CONTENTS{
    TCPA_SYMMETRIC_KEY sessionKey;
    TCPA_DIGEST idDigest;
} TCPA_ASYM_CA_CONTENTS;
```

**Parameters**

Type	Name	Description
TCPA_NONCE	SessionKey	This SHALL be the session key used by the CA to encrypt the TCPA_IDENTITY_CREDENTIAL
TCPA_DIGEST	idDigest	This SHALL be the digest of the TPM identity public key that is being certified by the CA

#### 4.25.7 TCPA\_SYM\_CA\_ATTESTATION

***Start of informative comment:***

This structure returned by the Privacy CA with the encrypted identity credential.

***End of informative comment.***

**Definition**

```
typedef struct tdTCPA_SYM_CA_ATTESTATION {
    UINT32 credSize;
    TCPA_ALGORITHM_PARMS algorithm;
    [size_is(credSize)] BYTE* credential;
} TCPA_SYM_CA_ATTESTATION;
```

Type	Name	Description
UINT32	credSize	This SHALL be the size of the credential parameter
TCPA_ALGORITHM_PARMS	algorithm	This SHALL be the indicator and parameters for the symmetric algorithm
BYTE*	credential	This is the result of encrypting TPM_IDENTITY_CREDENTIAL using the session_key and the algorithm indicated by sym_alg_id and sym_alg_parameters

## 4.26 TCPA\_CHANGEAUTH\_VALIDATE

**Start of informative comment:**

This structure provides an area that will stores the new authorization data and the challenger's nonce.

**End of informative comment.****Definition**

```
typedef struct tdTCPA_CHANGEAUTH_VALIDATE {  
    TCPA_SECRET newAuthSecret;  
    TCPA_NONCE n1;  
} TCPA_CHANGEAUTH_VALIDATE;
```

**Parameters**

Type	Name	Description
TCPA_SECRET	newAuthSecret	This SHALL be the new authorization data for the target entity
TCPA_NONCE	n1	This SHOULD be a nonce, to enable the caller to verify that the target TPM is on-line.

## 4.27 TCPA\_MIGRATIONKEYAUTH

***Start of informative comment:***

This structure provides the proof that the associated public key has TPM Owner authorization to be a migration key.

***End of informative comment.*****Definition**

```
typedef struct tdTCPA_MIGRATIONKEYAUTH{
    TCPA_PUBKEY migrationKey;
    TCPA_DIGEST digest;
} TCPA_MIGRATIONKEYAUTH;
```

**Parameters**

Type	Name	Description
TCPA_PUBKEY	migrationKey	This SHALL be the public key of the migration facility
TCPA_DIGEST	digest	This SHALL be the digest value of the migration key and tpmProof

## 4.28 TCPA\_PROTOCOL\_ID

**Start of informative comment:**

This value identifies the protocol in use.

**End of informative comment.****Definition**

```
typedef UINT16 TCPA_PROTOCOL_ID;
```

**TCPA\_PROTOCOL\_ID Values**

Value	Event Name	Comments
0x0001	TCPA_PID_OIAP	The OIAP protocol. See 0
0x0002	TCPA_PID_OSAP	The OSAP protocol. See 5.2.3
0x0003	TCPA_PID_ADIP	The ADIP protocol. See 5.2.6
0X0003	TCPA_PID_ADCP	The ADCP protocol. See 5.6



## 4.29 TCPA\_ENTITY\_TYPE

**Start of informative comment:**

This specifies the types of entity that are supported by the TPM.

**End of informative comment.****Definition**

```
typedef UINT16 TCPA_ENTITY_TYPE;
```

**TCPA\_ENTITY\_TYPE Values**

Value	Event Name	Comments
0x0001	TCPA_ET_KEYSLLOT	The entity is a keyslot
0x0002	TCPA_ET_OWNER	The entity is the TPM Owner
0x0003	TCPA_ET_DATA	The entity is some data
0x0004	TCPA_ET_SRK	The entity is the SRK
0x0005	TCPA_ET_KEY	The entity is a key
0x0006	TCPA_ET_IDENTITY	The entity is a TPM Identity

## 4.30 TCPA\_STARTUP\_TYPE

***Start of informative comment:***

To specify what type of startup is occurring.

***End of informative comment.*****Definition**

```
typedef UINT16 TCPA_STARTUP_TYPE;
```

**TCPA\_ENTITY\_TYPE Values**

Value	Event Name	Comments
0x0001	TCPA_ST_CLEAR	The TPM is starting up from a clean state
0x0002	TCPA_ST_STATE	The TPM is starting up from a saved state

## 4.31 Command Ordinals

### *Start of informative comment:*

The command ordinals provide the index value for each command

### *End of informative comment.*

#define TPM_ORD_OIAP	1001
#define TPM_ORD_OSAP	1002
#define TPM_ORD_ChangeAuth	1004
#define TPM_ORD_TakeOwnership	1005
#define TPM_ORD_ChangeAuthAsymStart	1006
#define TPM_ORD_ChangeAuthAsymFinish	1007
#define TPM_ORD_Extend	1010
#define TPM_ORD_PcrRead	1011
#define TPM_ORD_Quote	1012
#define TPM_ORD_Seal	1013
#define TPM_ORD_Unseal	1014
#define TPM_ORD_DirWriteAuth	1015
#define TPM_ORD_DirRead	1016
#define TPM_ORD_UnBind	1021
#define TPM_ORD_CreateWrapKey	1022
#define TPM_ORD_CreateWrapKeyToPcr	1023
#define TPM_ORD_LoadKey	1024
#define TPM_ORD_EvictKey	1025
#define TPM_ORD_BackupKey	1026
#define TPM_ORD_LoadBackupKey	1027
#define TPM_ORD_GetPubKey	1028
#define TPM_ORD_CreateMigrationBlob	1030
#define TPM_ORD_MigrateMigrationBlob	1031
#define TPM_ORD_LoadMigrationBlob	1032
#define TPM_ORD_AuthorizeMigrationKey	1033
#define TPM_ORD_CreateMaintenanceArchive	1034
#define TPM_ORD_LoadMaintenanceArchive	1035
#define TPM_ORD_KillMaintenanceFeature	1036
#define TPM_ORD_HashAll	1040
#define TPM_ORD_HashInit	1041
#define TPM_ORD_HashUpdate	1042
#define TPM_ORD_HashFinal	1043
#define TPM_ORD_HMACAll	1044
#define TPM_ORD_HMACInit	1045
#define TPM_ORD_HMACUpdate	1046
#define TPM_ORD_HMACFinal	1047
#define TPM_ORD_CertifyKey	1048
#define TPM_ORD_Sign	1050
#define TPM_ORD_VerifySignature	1051
#define TPM_ORD_GetRandom	1060
#define TPM_ORD_StirRandom	1061

```

#define TPM_ORD_SelfTestFull          1070
#define TPM_ORD_SelfTestStartup       1071
#define TPM_ORD_CertifySelfTest       1072

#define TPM_ORD_Reset                 1100
#define TPM_ORD_OwnerClear            1101
#define TPM_ORD_DisableOwnerClear     1102
#define TPM_ORD_ForceClear            1103
#define TPM_ORD_DisableForceClear     1104

#define TPM_ORD_GetCapabilitySigned    1105
#define TPM_ORD_GetCapability         1106

#define TPM_ORD_OwnerSetDisable       1107
#define TPM_ORD_PhysicalEnable        1108
#define TPM_ORD_PhysicalDisable       1109

#define TPM_ORD_CreateEndorsementKeyPair 1200
#define TPM_ORD_MakeTPMIdentity        1201
#define TPM_ORD_ActivateTPMIdentity    1202

#define TPM_ORD_RecoverTPMIdentity     1210

#define TPM_ORD_GetAuditEvent          1220

#define TPM_ORD_GetOrdinalAuditStatus  1250
#define TPM_ORD_SetOrdinalAuditStatus  1251

```

```

//-----
// TSS ordinals

```

```

#define TSS_ORD_EncryptAll             5001
#define TSS_ORD_EncryptInit            5002
#define TSS_ORD_EncryptUpdate          5003
#define TSS_ORD_EncryptFinal           5004

#define TSS_ORD_DecryptAll             5005
#define TSS_ORD_DecryptInit            5006
#define TSS_ORD_DecryptUpdate          5007
#define TSS_ORD_DecryptFinal           5008

#define TSS_ORD_CollateIdentityRequest 5009

#define TSS_ORD_Bind                   5020
#define TSS_ORD_WrapKey                5021
#define TSS_ORD_WrapKeyToPcr           5022

#define TSS_ORD_LogExtendEvent          5100
#define TSS_ORD_GetExtendEvent          5101
#define TSS_ORD_GetExtendEventLog      5102
#define TSS_ORD_DisposeEventLog        5103

#define TSS_ORD_GetAuditLog            5104

```

## 5. Authorization and Ownership

### 5.1 Introduction

***Start of informative comment:***

The purpose of the authorization mechanism is to authenticate an owner and to authorize use of an entity. The basic premise is to prove knowledge of a shared secret. This shared secret is the authorization data.

Authorization data is available for the TPM Owner and each entity that the TPM controls. The authorization data for the TPM and the SRK are held within the TPM itself and the authorization data for other entities are held with the entity.

The TPM Owner authorization data allows the Owner to prove ownership of the TPM. Proving ownership of the TPM does not immediately allow all operations – the TPM Owner is not a “super user” and additional authorization data must be provided for each entity or operation that has protection.

For each operation that uses an entity, the requestor must present the authorization data for the entity.

The TPM treats knowledge of the authorization data as complete proof of ownership of the entity. No other checks are necessary. The requestor (any entity that wishes to execute a command on the TPM or use a specific entity) may have additional protections and requirements where he or she (or it) saves the authorization data; however, the TPM places no additional requirements.

There are two protocols to securely pass a proof of knowledge of authorization data from requestor to TPM; the “Object-Independent Authorization Protocol” (OI-AP) and the “Object-Specific Authorization Protocol” (OS-AP). The OI-AP supports multiple authorization sessions for arbitrary entities. The OS-AP supports an authentication session for a single entity and enables the confidential transmission of new authorization information. That new authorization information is inserted by the “Authorization Data Insertion Protocol” (ADIP) during the creation of an entity. The “Authorization Data Change Protocol” (ADCP) and the “Asymmetric Authorization Change Protocol” (AACP) allow the changing of the authorization data for an entity. The protocol definitions allow expansion of protocol types to additional TCPA required protocols and vendor specific protocols.

The protocols use a “rolling nonce” paradigm. This requires that a nonce from one side be in use only for a message and its reply. For instance, the TPM would create a nonce and send that on a reply. The requestor would receive that nonce and then include it in the next request. The TPM would validate that the correct nonce was in the request and then create a new nonce for the reply. This mechanism is in place to prevent replay attacks and man-in-the-middle attacks.

The basic protocols do not provide long-term protection of authorization data that is the hash of a password or other low-entropy entities. The TPM designer and application writer must supply additional protocols if protection of these types of data is necessary.

The design criterion of the protocols is to allow for ownership authentication, command and parameter authentication and prevent replay and man-in-the-middle attacks.

The passing of the authorization data, nonces and other parameters must follow specific guidelines so that commands coming from different computer architectures will interoperate properly.

***End of informative comment.***

All protected commands and entity authorizations requiring authorization MUST use the authorization data protocols.

The TPM MUST support the OI-AP and the OS-AP which enable proof of knowledge of authorization data while maintaining the secrecy of that authorization data.

The TPM MUST support the ADIP that inserts the authorization during entity creation.

The TPM MUST support the ADCP and AACP which allow for the changing of authorization data.

The TPM MUST support TPM\_Terminate\_Handle which forces the termination of a session.

The TPM MAY support additional protocols to authenticate, insert and change authorization data.

The TPM MUST support the ability to calculate a HMAC in order to verify authorization data independent of the source or transmission mechanism. The TPM MUST calculate the HMAC parameters from the IDL representation of the command. The TPM MUST NOT perform the HMAC calculation for a returning message when the authorization for the command fails.

If a capability has more than one authorization value, each authorization process MUST use all authenticated parameters in its HMAC calculation. For example, the capability 9.3.1TPM\_MakeIdentity requires authorization from both the TPM Owner and from the SRK owner. So the authentication information "TpmOwnerAuth" and "SrAuth" are each calculated over all parameters tagged as "AUTH" in the definition of TPM\_MakeIdentity.

## 5.2 Authorization protocols

### ***Start of informative comment:***

The TPM provides two protocols for authorizing the use of entities without revealing the authorization data on the network or the connection to the TPM. In both cases, the protocol exchanges nonce-data so that both sides of the transaction can compute a hash using shared secrets and nonce-data. Each side generates the hash value and can compare to the value transmitted. Network listeners cannot directly infer the authorization data from the hashed objects sent over the network.

The first protocol is the “Object-Independent Authorization Protocol” (OI-AP), which allows the exchange of nonces with a specific TPM. Once an OI-AP session is established, its nonces can be used to authorize the use any entity managed by the TPM. The session can live indefinitely until either party request the session termination. The TPM\_OIAP function starts the OI-AP.

The second protocol is the “Object Specific Authorization Protocol” (OS-AP)”. The OS-AP allows establishment of an authentication session for a single entity. The session creates nonces that can authorize multiple commands without additional session-establishment overhead, but is bound to a specific entity. The TPM\_OSAPStart function starts the OS-AP. The TPM\_OSAPStart specifies the entity to which the authorization is bound.

Most commands allow either form of authorization protocol. In general, however, the OI-AP is preferred – it is more generally useful because it allows usage of the same session to provide authorization for different entities. The OS-AP is, however, necessary for operations that set or reset authorization data.

OI-AP sessions were designed for reasons of efficiency; only one setup process is required for potentially many authorizations.

An OS-AP session is doubly efficient because only one setup process is required for potentially many authorization calculations and the entity authorization secret is required only once. This minimizes exposure of the authorization secret and can minimize human interaction in the case where a person supplies the authorization information. The disadvantage of the OS-AP is that a distinct session needs to be setup for each entity that requires authorization. The OS-AP creates an ephemeral secret that is used throughout the session instead of the entity authorization secret. The ephemeral secret can be used to provide confidentiality for the introduction of new authorization data during the creation of new entities. Termination of the OS-AP occurs in two ways. Either side can request session termination (as usual) but the TPM forces the termination of an OS-AP session after use of the ephemeral secret for the introduction of new authorization data.

For both the OS-AP and the OI-AP, session setup is independent of the commands that are authorized. In the case of OI-AP, the requestor sends the TPM\_OIAP command, and with the response generated by the TPM, can immediately begin authorizing object actions. The OS-AP is very similar, and starts with the requestor sending a TPM\_OSAPStart operation, naming the entity to which the authorization session should be bound.

Both session types use a “rolling nonce” paradigm. This means that the TPM creates a new nonce value each time the TPM uses the session for a HMAC calculation.

Note that some operations involve the use of two authorization elements (for example, UNSEAL requires the authorization data of the object itself and authorization data of the object’s parent). In this case, two separate sessions are required. It is not possible to use one session for both purposes.

### ***End of informative comment.***

### 5.2.1 OI-AP description

**Start of informative comment:**

The purpose of this section is to illustrate the OI-AP without regard to a specific command. OI-AP uses the TPM\_OIAP command to create the authorization session. The definition for TPM\_OIAP is:

```
TCPA_RESULT TPM_OIAP(
    [out] TCPA_AUTHHANDLE* authHandle,
    [out] TCPA_NONCE* n0);
```

Assume that a TPM user wishes to send command C1. This is an authorized command that operates on entity E1 and requires the use of the authorization protocol. The user must know the authorization data for entity E1 (secretE1) as this is the entity that requires authorization. The user needs secretE1 in order to be able to calculate authC1E1.

Let us assume for this example that the caller of C1 does not need to authorize the use of entity E1 for more than execution of C1. This use model points to the selection of the OI-AP as the authorization protocol.

The command has the following layout:

```
TCPA_RESULT C1(
    [in, out] TCPA_AUTH* authC1E1,
    [AUTH in] X1,
    [AUTH in] Y1,
    [in] Z1,
    [out] O1,
    [AUTH out] O2);
```

For the C1 command, the AuthC1E1 parameter provides the authorization to execute the command. The following table shows the commands executed, the parameters created and the wire formats of all of the information.

Caller	On the wire	Dir	TPM
Send TPM_OIAP	TPM_OIAP	→	<ul style="list-style-type: none"> <li>Create session</li> <li>Create Handle H0</li> <li>Associate session and H0</li> <li>Generate Nonce N0</li> <li>Save N0 with H0</li> </ul>
Save H0 and N0	N0, H0	←	Returns
<ul style="list-style-type: none"> <li>Generate N1</li> <li>Compute AuthC1E1.digest = HMAC (SecretE1, C1, N0, N1, X1, Y1)</li> <li>AuthC1E1.nonce = N1</li> </ul>			
Send C1 (Ordinal TPM_ORD_C1)	TPM_ORD_C1, AuthC1E1, X1, Y1, Z1	→	<ul style="list-style-type: none"> <li>TPM retrieves SecretE1 (E1 and secretE1 must have been previously loaded)</li> <li>The TPM knows that AuthC1E1 refers to the X1 and Y1 parameters by the definition of C1.</li> <li>Verify AuthC1E1.authHandle points to a valid session, mismatch returns TPM_E_INVALIDAUTH</li> </ul>



			<ul style="list-style-type: none"> <li>Retrieve N0 from session</li> <li>HM = HMAC(SecretE1, C1, N0, N1, X1, Y1)</li> <li>Compare HM to AuthC1E1.digest. If they do not compare return with TPM_E_INVALIDAUTH</li> <li>Execute C1 and create return code (RC)</li> <li>Generate N2 to replace N0 in session</li> <li>Set AuthC1E1.digest = HMAC (SecretE1, C1, N2, N1, RETURNCODE, O2)</li> </ul>
<ul style="list-style-type: none"> <li>Save N2</li> <li>HM = HMAC(SecretE1, C1, N2, N1, RETURNCODE, O2)</li> <li>Compare HM to AuthC1E1.digest. This verifies RC and output parameters</li> </ul>	C1, AuthC1E1, N2, RETURNCODE, O1, O2	←	<ul style="list-style-type: none"> <li>Return result</li> <li>If AuthC1E1.continueuse is FALSE then destroy session</li> </ul>

Suppose now that the TPM user wishes to send command C2 using the same session. This is an authorized command that operates on entity E2. The user must know the authorization data for entity E2 (secretE2) as this is the entity that requires authorization. The user needs secretE1, in order to be able to calculate authC2E2.

The command has the following layout:

```
TCPA_RESULT C2(
    [in, out] TCPA_AUTH* authC2E2,
    [AUTH in] X2,
    [AUTH in] Y2,
    [in] Z2,
    [out] O3,
    [AUTH out] O4);
```

The following table shows the commands executed, the parameters created and the wire formats of all of the information.

Caller	On the wire	Dir	TPM
Send C2 (Ordinal TPM_ORD_C2)	TPM_ORD_C2, AuthC2E2, X2, Y2, Z2	→	<ul style="list-style-type: none"> <li>TPM retrieves SecretE2 (E2 and secretE2 must have been previously loaded)</li> <li>The TPM knows that AuthC2E2 refers to the X2 and Y2 parameters by the definition of C2.</li> <li>Verify AuthC2E2.authHandle points to a valid session, mismatch returns TPM_E_INVALIDAUTH</li> <li>Retrieve N0 from session</li> <li>HM = HMAC(SecretE2, C1, N2, N3, X2, Y2)</li> <li>Compare HM to AuthC2E2.digest. If they do not compare return with</li> </ul>

			TPM_E_INVALIDAUTH <ul style="list-style-type: none"> <li>• Execute C2 and create return code (RC)</li> <li>• Generate N4 to replace N2 in session</li> <li>• Set AuthC1E1.digest = HMAC ( SecretE2, C2, N4, N3, RETURNCODE, O4)</li> </ul>
<ul style="list-style-type: none"> <li>• Save N4</li> <li>• HM = HMAC( SecretE2, C2, N4, N3, RETURNCODE, O4)</li> <li>• Compare HM to AuthC2E2.digest. This verifies RC and output parameters</li> </ul>	C2, AuthC2E2, N4, RETURNCODE, O3, O4	←	<ul style="list-style-type: none"> <li>• Return result</li> <li>• If AuthC2E2.continueuse is FALSE then destroy session</li> </ul>

The TPM user could then use the session for further authorization sessions. Suppose, however, that the TPM user no longer requires the session. The user therefore issues a TPM\_Terminate\_Handle command to the TPM. The definition for Terminate\_Handle is:

```
TCPA_RESULT TPM_Terminate_Handle(
    [in] TCPA_PROTOCOL_ID protID,
    [in] UINT32 handle);
```

so the command is:

```
TPM_Terminate_Handle (
    PID_OI-AP>,
    H0);
```

In response, the TPM invalidates the session's handle and terminates the session's thread (releases all resources allocated to the session).

*End of informative comment.*

## 5.2.2 TPM\_OIAP

### IDL Definition

```
TCPA_RESULT TPM_OIAP(
    [in] TCPA_PROTOCOL_ID ProtocolID,
    [out] TCPA_AUTHHANDLE* AuthHandle,
    [out] TCPA_NONCE* n0);
```

### Type

TCPA protected capability.

### Parameters

Type	Name	Description
TCPA_PROTOCOL_ID	ProtocolID	The protocol in use MUST be TCPA_PID_OIAP.

TCPA_AUTHHANDLE	authHandle	Handle that TPM creates that points to the authorization state. The value is TPM specific and has no meaning except to identify the session.
TCPA_NONCE	N0	Nonce generated by TPM and associated with session.

### Actions

The TPM\_OIAP command allows the creation of an authorization handle and the tracking of the handle by the TPM. The TPM generates the handle and nonce.

The TPM has an internal limit as to the number of handles that may be open at one time, so the request for a new handle may fail if there is insufficient space available.

Internally the TPM will do the following:

1. TPM receives command.
2. TPM generates new handle and reserve space to save protocol identification, both nonces and any other information the TPM needs to manage the session.
3. TPM generates nonce N0.

On each subsequent use of the OIAP session the TPM MUST generate a new nonce value.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_SIZE	There are too many open auth handles
TCPA_FAIL	A critical internal error occurred

### 5.2.3 Authorization using an OI-AP session

#### ***Start of informative comment:***

This section describes the authorization-related actions of a TPM when it receives a command that has been authorized with the OI-AP protocol.

Many commands use OI-AP authorization. The following description is therefore necessarily abstract.

#### ***End of informative comment.***

### Actions

On reception of a command with ordinal C1 that uses an authorization session, the TPM SHALL perform the following actions:

1. The TPM MUST retrieve the secret authorization data (SecretE, say) of the target entity. The entity and its secret must have been previously loaded into the TPM.
2. The TPM MUST verify that the authorization handle (H, say) referenced in the command points to a valid session. If it does not, the TPM returns the error code TPM\_E\_INVALIDAUTH.
3. The TPM SHALL retrieve the latest version of the caller's nonce (N1, say) from the command.
4. The TPM SHALL retrieve the latest version of the TPM's nonce (N0, say) from the command.
5. The TPM SHALL retrieve the authenticated parameters (X, say) from the command.
6. The TPM performs a HMAC calculation (HM=HMAC[SecretE, C1, N0, N1, X], say)

7. The TPM SHALL compare HM to the authorization value received in the command. If they are different, the TPM returns the error code TPM\_E\_INVALIDAUTH. Otherwise, the TPM executes command C1 which produces an output (O, say) that requires authentication and uses a particular return code (RC, say).
8. The TPM SHALL generate the latest version of its nonce (N2, say).
9. The TPM creates a digest to authenticate the return values and return codes (ReturnDigest = HMAC [SecretE1, C1, N2, N1, RC, O])
10. The TPM returns the ReturnDigest to the caller along with C1, N2, RC, O and any other outputs that do not require authentication.
11. The TPM SHALL retrieve the continue flag from the received command. If the flag is FALSE, the TPM SHALL terminate the session and destroy the thread associated with handle H.

## 5.2.4 OS-AP Description

### **Start of informative comment:**

The OS-AP command creates an ephemeral secret to authenticate a session.

The purpose of this section is to illustrate the OS-AP without regard to a specific command. OS-AP uses the TPM\_OSAP command to create the authorization session. The definition for TPM\_OSAP is:

```
TCPA_RESULT TPM_OSAP(
    [in] TPCA_KEY_SLOT keySlotK,
    [in] TPCA_NONCE s1,
    [out] TPCA_NONCE* n0,
    [out] TPCA_NONCE* s0);
```

Assume that a TPM user wishes to send command C1. This is an authorized command that operates on the entity loaded into KeySlot K and requires the use of the authorization protocol. The user must know the authorization data for the entity E1 (secretE1) that is loaded in KeySlot K. The user needs secretE1 in order to be able to calculate authC1E1.

Let us assume for this example that the caller needs to authorize the use of KeySlot K for execution of more than just C1. This use model points to the selection of the OS-AP as the authorization protocol.

The command has the following layout:

```
TCPA_RESULT C1(
    [in, out] TPCA_AUTH AuthC1E1,
    [AUTH, in] X1,
    [AUTH, in] Y1,
    [in] Z1,
    [out] O1,
    [AUTH, out] O2);
```

For the C1 command, the AuthC1E1 parameter provides the authorization to execute the command. The following table shows the commands executed, the parameters created and the wire formats of all of the information. The TPM\_OSAP must have created the session using the entity E1 pointed to by keySlot K.

Caller	On the wire	Dir	TPM
• Generate S1			
Send TPM_OSAP	TPM_ORD_OSAP, keySlotK, S1	→	<ul style="list-style-type: none"> <li>• Create Handle H0</li> <li>• Save S1</li> <li>• Generate Nonces S0 and N0</li> <li>• Compute SharedSecret = HMAC(secretE1, S0, S1)</li> <li>• Set key slot indicator that OSAP now running</li> </ul>
Save H0, S0 and N0	H0, N0, S0	←	Returns
• SharedSecret = HMAC(secretE1, S0, S1)			
• Generate N1			
• AuthC1E1.blob = HMAC(SharedSecret, C1, N0, N1, X1, Y1)			
Send C1	TPM_ORD_C1, X1, Y1, Z1, AuthC1E1, N1	→	Save N1
			<ul style="list-style-type: none"> <li>• Validate AuthC1E1.blob = HMAC(SharedSecret, N0, N1, X1, Y1)</li> </ul>

			<ul style="list-style-type: none"> <li>Generate N2</li> <li>Execute C1</li> <li>AuthC1E1.blob = HMAC (SharedSecret, C1, N2, N1, RETURNCODE ,O2)</li> </ul>
<ul style="list-style-type: none"> <li>Save N2</li> <li>Validate AuthC1E1.blob = HMAC( SharedSecret, C1, N2, N1, RETURNCODE, O2)</li> </ul>	RETURNCODE, AuthC1E1, C1, N2, O1, O2	←	Returns

Suppose now that the TPM user wishes to send command C2 using the same session. The command has the following layout:

```
TCPA_RESULT C2(
    [in, out] TCPA_AUTH* authC2E1,
    [AUTH in] X2,
    [AUTH in] Y2,
    [in] Z2,
    [out] O3,
    [AUTH out] O4);
```

The following table shows the commands executed, the parameters created and the wire formats of all of the information.

Caller	On the wire	Dir	TPM
<ul style="list-style-type: none"> <li>Generate N3</li> <li>AuthC2E1.blob = HMAC (SharedSecret, C2, N2, N3, X2, Y2)</li> </ul>			
Send C2	TPM_ORD_C2, X2, Y2, Z2, AuthC2E1, N3	→	Save N3
			<ul style="list-style-type: none"> <li>Validate AuthC2E1.blob = HMAC ( SharedSecret, N2, N3, X2, Y2)</li> <li>Generate N4</li> <li>Execute C2</li> <li>AuthC2E1.blob = HMAC (SharedSecret, C2, N4, N3, RETURNCODE ,O4)</li> </ul>
<ul style="list-style-type: none"> <li>Save N4</li> <li>Validate AuthC2E2.blob = HMAC( SharedSecret, C2, N4, N3, RETURNCODE, O4)</li> </ul>	RETURNCODE, AuthC2E2, C2, N4, O3, O4	←	Returns

The TPM user could then use the session for further authorization sessions. Suppose, however, that the TPM user no longer requires the session. The user therefore issues a TPM\_Terminate\_Handle command to the TPM. The definition for Terminate\_Handle is:

```
TCPA_RESULT TPM_Terminate_Handle(
    [in] TCPA_PROTOCOL_ID protID,
    [in] UINT32 handle);
```

so the command is:

```
TPM_Terminate_Handle (
    PID_OS-AP,
    H0 );
```

In response, the TPM invalidates the session's handle and terminates the session's thread (releases all resources allocated to the session).

***End of informative comment.***

### 5.2.5 TPM\_OSAP

**Start of informative comment:**

The TPM\_OSAP command creates the authorization handle, the shared secret and generates N0 and S0.

**End of informative comment.**

**IDL Definition**

```
TCPA_RESULT TPM_OSAP(
    [in] TPCA_PROTOCOL_ID ProtocolID,
    [in] TPCA_ENTITY_TYPE entityType,
    [in] UINT32 entityValue,
    [in] TPCA_NONCE s1,
    [out] TPCA_AUTHHANDLE* authhandle,
    [out] TPCA_NONCE* n0,
    [out] TPCA_NONCE* s0);
```

**Type**

TCPA protected capability.

**Parameters**

Type	Name	Description
TCPA_PROTOCOL_ID	ProtocolID	The protocol in use MUST be TPCA_PID_OSAP.
TCPA_ENTITY_TYPE	entityType	The slot where the private key of the target entity is loaded. See
UINT32	entityValue	The selection value based on entityType
TCPA_NONCE	S1	The nonce generated by the caller as part of the shared secret
TCPA_AUTHHANDLE*	authHandle	Authorization structure that contains nonces and handle
TCPA_NONCE	N0	The even numbered nonce that the caller will use in the HMAC calculation
TCPA_NONCE	S0	The even numbered nonce that the caller will use to create the shared secret

**Actions**

The TPM\_OSAP command allows the creation of an authorization handle and the tracking of the handle by the TPM. The TPM generates the handle and the N0 and S0 nonces.

The TPM has an internal limit on the number of handles that may be open at one time, so the request for a new handle may fail if there is insufficient space available.

The TPM\_OSAP allows the binding of an authorization to a specific entity. This allows the caller to continue to send in authorization data for each command but not have to request the information or cache the actual authorization data.

Internally the TPM will do the following:

1. TPM receives command.
2. TPM generates new handle and reserves space to save protocol identification, shared secret, both nonces and any other information the TPM needs to manage the session.



3. TPM generates nonces N0 and S0.
4. TPM generates shared secret HMAC (authorization data, S0, S1) and saves secret in session area.
5. TPM fills in the TCPA\_AUTH fields and returns.

### Descriptions

#### entityType = TCPA\_ET\_KEYSLOT

The entity to authorize is a key slot. entityValue contains the key slot where the key is loaded. Key slot 0 identifies the SRK.

#### entityType = TCPA\_ET\_OWNER

This value indicates that the entity is the TPM owner. entityValue is ignored.

### Usage

On each subsequent use of the OSAP session the TPM MUST generate a new nonce value.

The TPM MUST ensure that OS-AP shared secret is only available while the OS-AP session is valid.

### Termination

The session MUST terminate upon any of the following conditions:

- The entity is unloaded. For keys this occurs when another key is loaded into the slot.
- The entity has a change authorization performed on it.
- The session is used in a TPM\_ChangeAuth command.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_SIZE	There are too many open auth handles
TCPA_FAIL	A critical internal error occurred

## 5.2.6 Authorization using an OS-AP session

### *Start of informative comment:*

This section describes the authorization-related actions of a TPM when it receives a command that has been authorized with the OS-AP protocol.

Many commands use OS-AP authorization. The following description is therefore necessarily abstract.

### *End of informative comment*

### Actions

On reception of a command with ordinal C1 that uses an authorization session, the TPM SHALL perform the following actions:

1. The TPM MUST retrieve the shared secret (Shared, say) of the target entity.
2. The TPM MUST verify that the authorization handle (H, say) referenced in the command points to a valid session. If it does not, the TPM returns the error code TPM\_E\_INVALIDAUTH.
3. The TPM SHALL retrieve the latest version of the caller's nonce (N1, say) from the command.
4. The TPM SHALL retrieve the latest version of the TPM's nonce (N0, say) from the command.
5. The TPM SHALL retrieve the authenticated parameters (X, say) from the command.
6. The TPM performs a HMAC calculation (HM=HMAC[Shared, C1, N0, N1, X], say)

7. The TPM SHALL compare HM to the authorization value received in the command. If they are different, the TPM returns the error code TPM\_E\_INVALIDAUTH. Otherwise, the TPM executes command C1 which produces an output (O, say) that requires authentication and uses a particular return code (RC, say).
8. The TPM SHALL generate the latest version of its nonce (N2, say).
9. The TPM creates a digest to authenticate the return values and return codes (ReturnDigest = HMAC [Shared, C1, N2, N1, RC, O])
10. The TPM returns the ReturnDigest to the caller along with N2, RC, O and any other outputs that do not require authentication.
11. The TPM SHALL retrieve the continue flag from the received command. If the flag is FALSE, the TPM SHALL terminate the session and destroy the thread associated with handle H.
12. If the shared secret was used to provide confidentiality for data in the received command, the TPM SHALL terminate the session and destroy the thread associated with handle H.



### 5.3 TPM\_Terminate\_Handle

**Start of informative comment:**

This allows the TPM manager to clear out information in a session handle

**End of informative comment.****IDL Definition**

```
TCPA_RESULT TPM_Terminate_Handle(  
    [in] TCPA_PROTOCOL_ID protID,  
    [in] UINT32 handle);
```

**Type**

TCPA protected capability.

**Parameters**

Type	Name	Description
TCPA_PROTOCOL_ID	protID	This SHALL indicate what type of handle to terminate.
UINT32	handle	This SHALL be the handle to terminate.

**Actions**

The TPM SHALL terminate the session and destroy all data associated with the session indicated.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_FAIL	A critical internal error occurred

## 5.4 ADIP – Creating a New Entity

### *Start of informative comment:*

The creation of the authorization data is the responsibility of the entity owner. He or she may use whatever process he or she wishes. The transmission of the authorization data from the owner to the TPM requires confidentiality and integrity. The encryption of the authorization data meets these requirements. The confidentiality and integrity requirements assume the insertion of the authorization data occurs over a network. While local insertions of the data would not require these measures, the protocol is established to be consistent with both local and remote insertions.

When the requestor is sending the authorization data to the TPM, the command to load the data requires the authorization of the entity owner. For example, to create a new TPM ID and set its authorization data requires the authorization data of the TPM Owner.

The confidentiality of the transmission comes from the encryption of the authorization data, and the integrity comes from the ability of the owner to verify that the authorization is being sent to a TPM and that only a specific TPM can decrypt the data.

The mechanism uses the following features of the TPM, OS-AP and HMAC.

- The creation of a new entity requires the authorization of the entity owner. When the requestor starts the creation process, the creator must use OS-AP.
- The creator encrypts the new authorization data using the shared secret from the OS-AP mechanism as a one-time pad with XOR and then sends this encrypted data along with the creation request to the TPM.
- The TPM decrypts the authorization data using the OS-AP shared secret, creates the new entity and sends the reply back to the creator using the new authorization data as the secret value of the HMAC.

The creator believes that the OS-AP creates a shared secret known only to the creator and the TPM. The TPM believes that the creator is the entity owner by their knowledge of the parent entity authorization data. The creator believes that the process completed correctly and that the authorization data is correct because the HMAC will only verify with the OS-AP secret.

The ADIP allows for the creation of new entities and the secure insertion of the new entity authorization data. The transmission of the new authorization data uses encryption with the key being a shared secret of an OS-AP session.

The OS-AP session must be created using the owner of the new entity.

We want to send command C3 to create a new entity, where the command structure is:

```
TCPA_RESULT C3(
    [in, out] TCPA_AUTH AuthY,
    [AUTH, in] NEWAUTH,
    [AUTH, in] X,
    [AUTH, in] Y,
    [in] Z,
    [out] O1,
    [AUTH, out] O2)
```

We assume that Entity Y is the parent for the new entity (TPM Owner if MakeTPMIdentity, otherwise a key).

**NewAuth** is the auth data for the new entity encrypted (XOR) using the shared secret created in TPM\_OSAPStart.

Caller	On the wire	Dir	TPM
• Generate S1			
Send TPM_OSAP	TPM_ORD_OSAP, pubKey, S1	→	<ul style="list-style-type: none"> <li>• Create Handle H0</li> <li>• Save S1</li> <li>• Generate Nonces S0 and N0</li> <li>• Compute SharedSecret = HMAC (secretPubKey, S0, S1)</li> <li>• Set key slot indicator that OSAP now running</li> </ul>
Save H0, S0 and N0	H0, N0, S0	←	Returns
• SharedSecret = HMAC(secretY, S0, S1)			
• Generate N1			
<ul style="list-style-type: none"> <li>• Encrypt NEWAUTH using SharedSecret as key</li> <li>• AuthY.blob = HMAC (SharedSecret, C3, N0, N1, X, Y, NEWAUTH)</li> </ul>			
Send C3	TPM_ORD_C3, AuthY, NEWAUTH, X, Y, Z	→	Save N1
			<ul style="list-style-type: none"> <li>• Validate AuthY.blob = HMAC (SharedSecret, C3, N0, N1, X, Y, NEWAUTH)</li> <li>• Decrypt NEWAUTH using SharedSecret as key</li> <li>• Execute C3</li> <li>• Generate N2</li> <li>• AuthY.blob = HMAC (SharedSecret, C3, N2, N1, RETURNCODE, O2)</li> <li>• Destroy SharedSecret</li> </ul>
Save N2	RETURNCODE, SessionAuth, N2	←	Returns
<ul style="list-style-type: none"> <li>• Validate AuthY.blob = HMAC( SharedSecret, C3, N2, N1, RETURNCODE, O2)</li> <li>• Save new entity</li> </ul>			

**End of informative comment.**

The TPM MUST enable ADIP by using the OS-AP. The TPM MUST encrypt the authorization data for the new entity by performing an XOR using the shared secret created by the OS-AP.

The TPM MUST destroy the OS-AP session whenever a new entity is created.

## 5.5 ADCP - Changing Authorization Data

### ***Start of informative comment:***

All entities from the Owner to the SRK to individual keys and data blobs have authorization data. This data may need to change at some point in time after the entity creation. The ADCP allows the entity owner to change the authorization data (for a wrapped key, the entity owner is its parent key).

A requirement is that the Owner must remember the old authorization data. The only mechanism to change the authorization data when the entity Owner forgets the current value is to delete the entity and then recreate it.

To protect the data from exposure to eavesdroppers or other attackers, the authorization data uses the same encryption mechanism in use during the ADIP.

Changing authorization data requires opening two authentication handles. The first handle authenticates the entity Owner (or parent) and the right to load the entity. This first handle is an OS-AP and supplies the data to encrypt the new authorization data according to the ADIP protocol. The second handle can be either an OI-AP or an OS-AP, it authorizes access to the entity for which the authorization data is to be changed.

The authorization data in use to generate the OS-AP shared secret must be the authorization data of the parent of the entity to which the change will be made.

In the case of TPM identities, the parent ("owner") of all TPM identities is the SRK. Unfortunately, the SRK may have a well known value as its authorization data. To avoid the problem of encrypting information with a well known value, all TPM identity authorization data changes require the first handle OS-AP to be setup using the TPM Owner authorization data. This is necessary to ensure that the new authdata is not introduced XORing with an OS-AP shared secret based on the SRK authdata, which may be a well known value.

Similarly, when changing the authorization data for the SRK, the first handle OS-AP must be setup using the TPM Owner authorization data. This is because the SRK does not have a parent, per se.

To summarize: only a TPM owner can change TPM identity or SRK authorization data

### ***End of informative comment.***

Changing authorization data for the TPM requires authorization of the current TPM Owner.

Changing authorization data for the SRK requires authorization of the TPM Owner.

Changing authorization data for a TPM Identity requires authorization of the TPM Owner.

All other entities require authorization of the parent entity.

## 5.6 TPM\_ChangeAuth

### *Start of informative comment:*

The TPM\_ChangeAuth command allows the owner of an entity to change the authorization data for the entity.

### *End of informative comment.*

### IDL Definition

```
TCPA_RESULT TPM_ChangeAuth(
    [in, out] TCPA_AUTH* ParentAuth,
    [in, out] TCPA_AUTH* BlobAuth,
    [AUTH, in] TCPA_ENTITY_TYPE TargetType,
    [AUTH, in] TCPA_PROTOCOL_ID ProtocolID,
    [AUTH, in] UINT32 BlobSize,
    [AUTH, in] UINT32 MaxNewBlobSize,
    [AUTH, in] TCPA_ENCAUTH NewAuth,
    [AUTH, in] TCPA_KEY_SLOT ParentRef,
    [AUTH, in, out] UINT32* NewBlobSize,
    [AUTH, in, size_is(BlobSize)] BYTE* Blob,
    [AUTH, out, size_is(*NewBlobSize)] BYTE* NewBlob);
```

### Type

TCPA protected capability; user must provide authorizations for the entity pointed to by parentRef and blob.

### Parameters

Type	Name	Description
TCPA_AUTH	ParentAuth	Authorization structure that contains authorization data for the parent key.
TCPA_AUTH	BlobAuth	Authorization structure that contains authorization data for the Entity in parameter blob.
TCPA_ENTITY_TYPE	TargetType	What entity to change the authorization data for
TCPA_PROTOCOL_ID	ProtocolID	The ownership protocol in use.
UINT32	BlobSize	Size of the incoming blob.
UINT32	MaxNewBlobSize	The maximum size of the data buffer for the out going blob.
TCPA_ENCAUTH	NewAuth	This SHALL be the encrypted new authorization data.
TCPA_KEY_SLOT	ParentRef	Reference to the parent of the blob
UINT32*	NewBlobSize	The actual size of the out going blob. Must be smaller than maxNewBlobSize.
BYTE*	Blob	The entity who's authorization needs changing.
BYTE*	NewBlob	The new blob.



## Actions

This section defines the TPM\_PID\_ADCP protocol. Additional protocols would have different requirements.

A TPM MUST support TPM\_PID\_ADCP.

The TPM Owner and the SRK are Internal entities. All others (wrapped key, sealed data) are External entities.

### If the TargetType is T CPA\_ET\_OWNER or T CPA\_ET\_SRK

The session pointed to by parentAuth MUST be T CPA\_PID\_O SAP using the TPM Owner authorization data. The newAuth parameters MUST point to the new authorization data, and are protected according to the ADIP protocol (XORed with the T CPA\_PID\_O SAP shared secret that is based on the TPM Owner's authorization data)

The TPM MUST ignore all other parameters.

The TPM MUST enforce the destruction of the parentAuth session upon completion of this command (successful or unsuccessful).

### If TargetType is T CPA\_ET\_DATA, T CPA\_ET\_KEY or T CPA\_ET\_IDENTITY

The session pointed to by parentAuth MUST be an T CPA\_PID\_O SAP using the Parent Entity's authorization data. The newAuth parameter MUST point to the new authorization data, and is protected according to the ADIP protocol (XORed with the T CPA\_PID\_O SAP shared secret that is based on the parent entity's authorization data).

When TargetType is T CPA\_ET\_IDENTITY, the parentAuth session MUST be a T CPA\_PID\_O SAP using the TPM Owner authorization data.

The TPM MUST validate the command using the authorization data in the parentAuth parameter. The parentRef parameter provides the identification of the parent.

After validation the TPM attempts to decrypt the blob using the key pointed to by ParentRef. The TPM then attempts to validate that the decrypted blob is a valid structure. Then the TPM authorizes the use of the decrypted entity using the authorization in the blobAuth parameter.

Failure to validate either of these entities results in the TPM returning an error code to the caller.

The TPM then decrypts the newAuth parameter (using the authorization data of the entity pointed to by ParentRef) and replaces the authorization data in the decrypted blob with the new decrypted value. The TPM then encrypts the blob using the parent and places the result in the newBlob area. The newAuth area is encrypted using the ADIP mechanism.

The TPM MUST enforce the destruction of the T CPA\_PID\_O SAP session upon completion of this command (successful or unsuccessful).

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_FAIL	A critical internal error occurred

## 5.7 Asymmetric Authorization Change Protocol

### *Start of informative comment:*

This asymmetric change protocol allows the entity owner to change entity authorization, under the parent's execution authorization, to a value of which the parent has no knowledge.

In contrast, the TPM\_ChangeAuth command uses the parent entity authorization data to create the shared secret that encrypts the new authorization data for an entity. This creates a situation where the parent entity ALWAYS knows the authorization data for entities in the tree below the parent. There may be instances where this knowledge is not a good policy.

This asymmetric change process requires two commands and the use of an OI-AP session. The OI-AP session must hold the state for an ephemeral key created solely for the encryption of the authorization data.

### *End of informative comment.*

### 5.7.1 TPM\_ChangeAuthAsymStart

#### *Start of informative comment:*

The TPM\_ChangeAuthAsymStart starts the process of changing authorization for an entity. It sets up an OI-AP session that must be retained for use by its twin TPM\_ChangeAuthAsymFinish command.

TPM\_ChangeAuthAsymStart creates a temporary asymmetric public key "tempkey" to provide confidentiality for new authorization data to be sent to the TPM. TPM\_ChangeAuthAsymStart certifies that tempkey was generated by a genuine TPM, by generating a certifyInfo structure that is signed by a TPM identity. The owner of that TPM identity must cooperate to produce this command, because TPM\_ChangeAuthAsymStart requires authorization to use that identity.

It is envisaged that tempkey and certifyInfo are given to the owner of the entity whose authorization is to be changed. That owner uses certifyInfo and a TPM\_IDENTITY\_CREDENTIAL to verify that tempkey was generated by a genuine TPM. This is done by verifying the TPM\_IDENTITY\_CREDENTIAL using the public key of a CA, verifying the signature on the certifyInfo structure with the public key of the identity in TPM\_IDENTITY\_CREDENTIAL, and verifying tempkey by comparing its digest with the value inside certifyInfo. The owner uses tempkey to encrypt the desired new authorization data and inserts that encrypted data in a TPM\_ChangeAuthAsymFinish command, in the knowledge that only a TPM with a specific identity can interpret the new authorization data.

#### *End of informative comment.*

### IDL Definition

```
TCPA_RESULT TPM_ChangeAuthAsymStart(
    [in, out] TPCA_AUTH* IDAuth,
    [AUTH, in] TPCA_KEY_SLOT idSlot,
    [AUTH, in] TPCA_KEY_SLOT ephSlot,
    [AUTH, in] UINT32 maxSigSize,
    [AUTH, in, out] UINT32* sigSize,
    [AUTH, in, out] TPCA_KEY* tempkey,
    [AUTH, out] TPCA_CERTIFY_INFO* certifyInfo,
    [AUTH, out, size_is(*sigSize)] BYTE* signature);
```

### Type

TCPA protected capability; user must provide authorization for the identity in idSlot.

### Parameters

Type	Name	Description
TCPA_AUTH*	IDAuth	Authorization structure that contains authorization data for the TPM Identity. For this parameter, the Authorization Type MUST be OI-AP.
TCPA_KEY_SLOT	idSlot	This SHALL be the key slot where the identity is loaded. This MUST be a TPM identity key.
TCPA_KEY_SLOT	ephSlot	This SHALL indicate the key slot to hold the ephemeral key
UINT32	maxSigSize	This SHALL be the maximum size of the signature parameter
UINT32*	sigSize	This SHALL be the size of the signature parameter
TCPA_KEY*	tempKey	The input structure contains all parameters except pubkey and privkey (which are NULL), to specify the size and type of ephemeral key. The output structure also contains pubkey (the public part of the new ephemeral key). The privkey field in the output structure is NULL.
TCPA_CERTIFY_INFO	certifyInfo	This SHALL be the TCPA_CERTIFY_INFO structure that is signed.
BYTE*	signature	This SHALL be the signature on the certifyInfo parameter using the key in idSlot.

### Actions

- The TPM SHALL verify the authorization to use the TPM identity key held in idSlot. The TPM MUST verify that the key is a TPM identity key.
- The TPM SHALL validate the algorithm parameters for the key to create from the tempKey parameter. The minimum RSA key size MUST be 512 bits.
- The TPM SHALL create a new key from the tempKey parameters and associate the internal storage of this newly created key with the OI-AP session handle provided by IDAuth parameter.
- The TPM SHALL fill in the TCPA\_PUBKEY section of the tempKey parameter. The TPM MUST set the TCPA\_PRIVKEY section to null.
- The TPM SHALL fill in the TCPA\_CERTIFY\_INFO structure for the newly created key. This structure SHALL be returned in parameter certifyInfo. See below for field values.
- The TPM then performs a TPM\_Internal\_Signature (See 8.16.2) on the certifyInfo parameter using the key pointed to by idSlot. The resulting signed blob is returned in signature parameter.

### Field Descriptions for certifyInfo parameter

Name	Description
Version	TCPA version structure; section 4.5.
Keyflags.IsWrappedToPCR	This SHALL be set to FALSE.
Keyflags.Redirection	This SHALL be set to FALSE.
Keyflags.Migratable	This SHALL be set to FALSE.
Keyflags.Volatile	This SHALL be set to TRUE.
Keyflags.Migration	This SHALL be set to FALSE.

pcrList	This SHALL be an empty list.
typeOfKey	This SHALL be set to algorithm in use
typeTag	This SHALL reflect the newly created key algorithm information.
authDataUsage	This SHALL be set to TPM_AUTH_ALWAYS.
KeyUsage	This SHALL be set to TPM_KEY_AUTHCHANGE
DigestValue	This SHALL be set to NULL
pubDigest	This SHALL be the hash of the public key being certified.
Data	This SHALL be set to NULL

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_FAIL	A critical internal error occurred

### 5.7.2 TPM\_ChangeAuthAsymFinish

**Start of informative comment:**

The TPM\_ChangeAuth command allows the owner of an entity to change the authorization data for the entity. It must use the same OI-AP session as its twin TPM\_ChangeAuthAsymStart command.

The command requires the cooperation of the owner of the parent of the entity, since authorization must be provided to use that parent entity. The command requires knowledge of the existing authorization information and passes the new authorization information. The “BlobAuth” parameter proves knowledge of existing authorization information and new authorization information. The new authorization information “encNewAuth” is encrypted using the “tempKey” variable obtained via TPM\_ChangeAuthAsymStart.

A parent therefore retains control over a change in the authorization of a child, but is prevented from knowing the new authorization data for that child.

**End of informative comment.**

#### IDL Definition

```
TCPA_RESULT TPM_ChangeAuthAsymFinish(
    [in, out] TCPA_AUTH* ParentAuth,
    [AUTH, in] TCPA_ENTITY_TYPE TargetType,
    [AUTH, in] UINT32 BlobSize,
    [AUTH, in] UINT32 MaxNewBlobSize,
    [AUTH, in] UINT32 NewAuthSize,
    [AUTH, in] TCPA_KEY_SLOT ParentRef,
    [AUTH, in] TCPA_DIGEST BlobAuth,
    [AUTH, in, size_is(NewAuthSize)] BYTE* encNewAuth,
    [AUTH, in, size_is(BlobSize)] BYTE* Blob,
    [AUTH, in, out] UINT32* NewBlobSize,
    [AUTH, out] TCPA_DIGEST* ChangeProof,
    [AUTH, out, size_is(*newBlobSize)] BYTE* NewBlob);
```

#### Type

TCPA protected capability; caller must provide authorizations for the entity pointed to by parentRef and blob.

#### Parameters

Type	Name	Description
TCPA_AUTH	ParentAuth	Authorization structure that contains authorization data for the parent key.  The OI-AP session MUST be the same session as the one used in TPM_ChangeAuthAsymStart command that created the public key used to encrypt the contents of the NewAuth parameter.
TCPA_ENTITY_TYPE	TargetType	The type of the entity in “Blob” whose authorization is to be changed.
UINT32	BlobSize	Size of the incoming blob.
UINT32	MaxNewBlobSize	The maximum size of the data buffer for the outgoing blob.
UINT32	NewAuthSize	The size of the new authorization data.

TCPA_KEY_SLOT	ParentRef	Reference to the parent of the blob
TCPA_DIGEST	BlobAuth	A HMAC that links the old authorization data with the new authorization data. See below.
BYTE*	encNewAuth	The new authorization data structure encrypted under the public key associated with the OI-AP session.
BYTE*	Blob	The entity whose authorization needs changing.
UINT32*	NewBlobSize	The actual size of the out going blob.
TCPA_DIGEST*	ChangeProof	This SHALL contain a cryptographic proof that the authorization data changed.
BYTE*	NewBlob	The new blob.

### Actions

- The TPM SHALL validate that the ParentAuth parameter authorizes use of the key in parentRef.
- The TPM SHALL validate that the session in use by ParentAuth is managing an active and valid TPM\_KEY\_AUTHCHANGE key.
- The TPM SHALL decrypt the entity held in the Blob parameter and validate the structure of the decrypted entity.
- The TPM SHALL decrypt the encNewAuth blob using the private key of the TPM\_KEY\_AUTHCHANGE key pair. The decrypted area contains a structure newAuth of type TCPA\_CHANGEAUTH\_VALIDATE.
- The TPM SHALL create blobverify by performing the following HMAC calculation: blobverify = HMAC(newAuth.newAuthSecret) using blob.currentAuth as the secret. Where the currentAuth is the current shared authorization secret and the newAuth.newAuthSecret area is the new shared authorization secret.
- The TPM SHALL compare the blobverify value with the BlobAuth parameter. The TPM SHALL indicate a failure if the values do not match.
- The TPM SHALL replace decryptedblob.authdata with newAuth.newAuthSecret.
- The TPM SHALL encrypt the decryptedblob structure using the appropriate wrap command with the key in ParentRef.
- The TPM SHALL create ChangeProof parameter by creating an HMAC. ChangeProof = HMAC(newAuth.newAuthSecret, newAuth.nonce1). Where the newAuth parameter is the new shared authorization secret and nonce1 is the nonce.
- The TPM MUST destroy the TPM\_KEY\_AUTHCHANGE key associated with the OI-AP session.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_FAIL	A critical internal error occurred

## 5.8 Authorization Data

### ***Start of informative comment:***

The authorization data is a 160-bit field that the TPM stores in a “shielded location,” which is an area where data is protected against interference and prying, independent of its form. The Owner has a copy of the data and protects the data using whatever mechanism the Owner wishes to use. The authorization data is a shared secret between the TPM and the Owner of the entity. There are no requirements as to what the 160 bit of data are. The assumption is that the data is a SHA-1 hash of a password or other data, but the data can be anything.

There will be a separate piece of authorization data for each entity. There is no requirement that each authorization data blob must be unique.

The TPM treats the authorization data as shielded data, an approach that requires that only TPM-protected capabilities access the authorization data. A further requirement is that the only use of the authorization data within the TPM is in the authorization process. No other use is permissible.

The protection of the backup mechanism is a type of authorization.

### ***End of informative comment.***

The TPM MUST reserve 160 bits for the authorization data. The TPM treats the authorization data as a blob. The TPM MUST keep the authorization data in a shielded location.

The TPM MUST enforce that the only usage in the TPM of the authorization data is to perform authorizations.

## 5.9 Nonces

### ***Start of informative comment:***

All of the authorization protocols require nonces to prevent replay and man-in-the-middle attacks. To further strengthen the use of the nonces a rolling-nonce paradigm requires the use of new nonces for each message and response.

The nonce values from the TPM must use the internal RNG. The nonce values from the requestor can use any source that provides information to the requestor. The highest value is obtained when the requestor also uses an RNG for the nonce values; however, there is no loss of security to the TPM if set values are in use. The requestor loses some protection when he or she (or it) uses set values.

In all descriptions of nonce usage in this section all odd nonce values come from the challenger, all even nonce values come from the TPM (0 is an even number for this definition).

The requestor is responsible for generating and sending the odd nonce value. The TPM will enforce that the odd nonce value changes for each request.

The TPM is responsible for the even nonce values. The TPM changes the value of the even nonce on each reply.

### ***End of informative comment.***

The requestor MUST provide a unique value in the nonce field of the authorization structure for each request.

The TPM MUST supply a new nonce value for each reply. The nonce value MUST come from the internal RNG. The TPM MUST enforce the validity of the returning nonce another command uses the authorization session.

## 5.10 Authorization Handle

***Start of informative comment:***

The TPM generates authorization handles to allow for the tracking of information regarding a specific authorization invocation.

The TPM saves information specific to the authorization, such as the nonce values, ephemeral secrets and type of authentication in use.

The TPM may create any internal representation of the handle that is appropriate for the TPM's design. The requestor always uses the handle in the authorization structure to indicate authorization structure in use.

The TPM must support a minimum of two concurrent authorization handles. The use of these handles is to allow the Owner to have an authorization active in addition to an active authorization for an entity.

To ensure garbage collection and the proper removal of security information, the requestor should terminate all handles. Termination of the handle uses the continue-use flag to indicate to the TPM that the handle should be terminated.

Termination of a handle instructs the TPM to perform garbage collection on all authorization data. Garbage collection includes the deletion of the ephemeral secret.

***End of informative comment.***

The TPM **MUST** support authorization handles. The TPM **MUST** support a minimum of two concurrent authorization handles.

The TPM **MUST** support authorization-handle termination. The termination includes garbage collection of authorization data.



## 5.11 HMAC Calculation

### ***Start of informative comment:***

The HMAC provides two pieces of information to the TPM: proof of knowledge of the authorization data and proof that the request arriving is authorized and has no modifications made to the command in transit.

The HMAC definition is for the HMAC calculation only. It does not specify the order or mechanism that transports the data from caller to actual TPM.

The creation of the HMAC is order dependent. Each command has specific items that are portions of the HMAC calculation. The actual calculation starts with the definition from RFC 2104.

RFC 2104 requires the selection of two parameters to properly define the HMAC in use. These values are the key length and the block size. This specification will use a key length of 20 bytes and a block size of 64 bytes. These values are known in the RFC as K for the key length and B as the block size.

The basic construct is

$$H(K \text{ XOR } \text{opad}, H(K \text{ XOR } \text{ipad}, \text{text}))$$

where

- H = the SHA1 hash operation
- K = the key or the authorization data
- XOR = the XOR operation
- opad = the byte 0x5C repeated B times
- B = the block length
- ipad = the byte 0x36 repeated B times
- text = the message information and any parameters from the command

### ***End of informative comment.***

The TPM MUST support the calculation of an HMAC according to RFC 2104.

The key size (K in RFC 2104) MUST be 20 bytes. The block size (B in RFC 2104) MUST be 64 bytes.

When a command has two HMAC calculations (i.e. it has two TCPA\_AUTH parameters) then BOTH calculations MUST use the same parameters and nonces. The only difference between the two calculations is the secret.

The order of the parameters is critical to the TPM's ability to recreate the HMAC. Not all of the fields are sent on the wire for each command for instance only one of the nonce values travels on the wire. The text field follows the following construction format:

- The authorization data for the entity to release. The authorization data is the shared secret that both sides of the conversation are trying to prove knowledge of.
- Command ordinal
- TCPA\_AUTH fields excluding the nonce value
- Even Nonce (from TPM)
- Odd Nonce (from requestor)
- On return the return code is included the calculation.
- Parameters from the command selected from the left of the definition. Those commands that are security sensitive have the AUTH decoration in the IDL fields.

### 5.11.1 HMAC Long Parameters

***Start of informative comment:***

A TPM implementation for reasons of speed or efficiency may wish to delay the transmission of a parameter to the TPM. To facilitate this ability the HMAC calculation does not occur on parameters that are longer than 20 bytes. Rather the caller performs a SHA-1 hash of the parameter and then incorporates the hash result in the HMAC calculation. The caller must transmit the parameter and the 20 byte hash result.

The requirement to perform the hash operation before use of the parameter allows a driver to delay transmission of a large parameter, validate HMAC and when required transmits the parameter.

***End of informative comment.***

The HMAC calculation **MUST** use a 20-byte hash value instead of the actual parameter value for all parameters with a length greater than 128 bytes.

The TPM **MUST** perform the hash operation and validate the HMAC calculation before using an individual parameter.

## 5.12 TPM Ownership

### ***Start of informative comment:***

The Owner of the TPM has the right to perform special operations. The process of taking ownership is the procedure whereby the Owner inserts a shared secret into the TPM. For all future operations, knowledge of the shared secret is proof of ownership. When the Owner wishes to perform one of the special operations then the Owner must use the authorization protocol to prove knowledge of the shared secret.

The TPM default state is to have no Owner.

The difficulty with ownership is inserting the shared secret in a secure manner. A design consideration is that the taking of ownership must be an operation that works securely over the network. The function must provide confidentiality and integrity to the messages sent to the TPM.

The function to insert the Owner must provide the following:

- Confidentiality. The shared secret (or authorization data) must remain confidential to all eavesdroppers that intercept any of the messages. The confidentiality comes from encrypting the shared secret using the TPM PUBEK. The Owner trusts that only the TPM has the PRIVEK that can decrypt the shared secret.
- Integrity. The TPM and the Owner must be able to determine the integrity of messages and responses to the function. The integrity checking does not have to occur at the instant of receiving a message. The Owner validates the integrity of the messages using the HMAC construct.
- Remoteness – the function must allow the Owner to take control across a network.
- Verifiability. The function allows the Owner to verify that he or she has truly taken control. The Owner verifies that the secret was successfully installed by verifying the HMAC response. Additional verification can occur by attempting to establish a Owner session.

The TPM\_TakeOwnership function inserts the Owner-authorization data and creates a new Storage Root Key (SRK). The TPM\_TakeOwnership function fails if there is already an Owner set for the TPM.

After inserting the authorization data, the TPM\_TakeOwnership function creates the SRK. By default SRK entities have no authorization data associated with them. The owner should set the SRK secret to null (by passing in a zero length) to set the SRK authorization data to null.

To validate that the operation completes successfully, the TPM HMACs the response to the TPM\_TakeOwnership function.

### ***End of informative comment.***

The TPM MUST ship with no Owner installed. The TPM MUST use the ownership-control protocol.

### 5.12.1 TPM\_TakeOwnership

#### IDL Definition

```
TCPA_RESULT TPM_TakeOwnership (
    [in, out] TCPA_AUTH* auth,
    [AUTH, in] UINT32 ProtocolID,
    [AUTH, in] TCPA_ENCAUTH EncOwnerAuth,
    [AUTH, in] TCPA_ENCAUTH EncSrkJAuth,
    [AUTH, in, out] TCPA_KEY* SrkJPub);
```

#### Type

TCPA protected capability; user must encrypt the values using the PUBKEY.

#### Parameters

Type	Name	Description
TCPA_AUTH*	auth	The authorization from the TPM Owner. There is no validation of in parameters, just validation on the return that the proper authorization data was used.
UINT32	ProtocolID	The ownership protocol in use. The default protocol for version 1.0 is TPM_PRT_OWNER.
TCPA_ENCAUTH	EncOwnerAuth	The encrypted Owner authorization data
TCPA_ENCAUTH	EncSrkJAuth	The encrypted Storage Root Key (SRK) authorization data
TCPA_KEY	SrkJPub	The input structure contains all parameters except pubkey and privkey (which are NULL), to specify the size and type of SRK. The output structure also contains pubkey (the public part of the SRK). The privkey field in the output structure is NULL.

#### Actions

The new owner **MUST** encrypt the Owner authorization data and the SRK authorization data using the PUBKEY. The endorsement key pair **MUST** be an RSA key so the encryption algorithm in use to encrypt these secrets is RSA.

If the TPM has a current owner then the TPM upon receipt of this command **SHALL** return the error code TCPA\_OWNER\_SET.

If the TPM has no current owner then the TPM upon receipt of this command **SHALL**:

1. Decrypt EncOwnerAuth using the PRIVEK to generate ProspectiveOwnerAuth.
2. Use the TCPA authorization protocol to verify that all input parameters tagged with AUTH have been sent by an entity that knows ProspectiveOwnerAuth. If any verification fails, abandon this process and do not return a value to the caller. Otherwise, continue with this process.
3. Store ProspectiveOwnerAuth as the Owner's authorization data.
4. Generate a new SRK. The SRK **MUST** be a 2048 bit RSA key.
5. Decrypt EncSrkJAuth using the PRIVEK and store the result as the SRK's authorization data.
6. Return the public part of the SRK to the caller.
7. Calculate an authenticated response using the new authorization data

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_OWNER_SET	There is already an Owner and we do not want a new one
TCPA_FAIL	A critical internal error occurred

## 6. Integrity Collection and Reporting

### 6.1 Introduction

***Start of informative comment:***

The TCPA Trusted Platform Support Services(TSS) provides mechanisms for cryptographically reporting the current hardware and software configuration of a computing device to local and remote Challengers. The TSS also provides a limited protected storage capability, which allows the Subsystem Owner to store an acceptable platform configuration, biometric data or other data that is available early in boot. System firmware or other software may use this storage capability to name Users qualified to log on, or acceptable boot configurations. TCPA specification does *not* define how this storage facility should be used.

The TSS also provides a facility whereby platform software or firmware may store secrets that are accessible only when the platform is in a defined configuration. This mechanism is known as *sealing*. The following sections describe and define the Trusted Platform Module (TPM)–protected operations that support integrity collection and reporting. The usage required in a TCPA-compliant PC platform is described in a separate document.

***End of informative comment.***

## 6.2 Platform Configuration Registers

### 6.2.1 Format and Properties

A Platform Configuration Register (PCR) consists of a 160-bit field that holds a cumulatively updated hash value and a 4byte status field. The PCR data structure **MUST** be a TCPA-shielded location. PCRs **SHOULD** be in volatile storage. The PCRs **MUST** be set to 0 before first use. This specification does not mandate the internal storage format.

A TPM implementation **MUST** provide eight or more independent PCRs. These PCRs are identified by index and **MUST** be numbered from 0 (that is, PCR<sub>0</sub> through PCR<sub>7</sub> are required for TCPA compliance). Vendors **MAY** implement more registers for general-purpose use. Extra registers **MUST** be numbered contiguously from 8 up to max – 1, where max is the maximum offered by the TPM.

The TCPA-protected capabilities that expose and modify the PCRs use a 32-bit index, indicating the maximum usable PCR index. However, TCPA reserves register indices  $2^{30}$  and higher for later versions of the specification. A TPM implementation **MUST NOT** provide registers with indices greater than or equal to  $2^{30}$ . The register index  $2^{32}-1$  is used as a wildcard identifier for TPM\_Seal and TPM\_Unseal; it does not identify an actual PCR. In this specification, the following terminology is used (although this internal format is not mandated).

### 6.2.2 Initialization

PCRs and the protected capabilities that operate upon them **MAY NOT** be used until power-on self-test (TPM POST) has completed. If TPM POST fails, the TPM\_Extend operation will fail; and, of greater importance, the TPM\_Quote operation and TPM\_Seal operations that respectively report and examine the PCR contents **MUST** fail. At the successful completion of TPM POST, all PCRs **MUST** be set to 0. Additionally, the UINT32 flags **MUST** be set to zero.

### 6.2.3 Authorized PCRs

A TPM **MUST** provide one Data Integrity Register (DIR). Implementations **MAY** provide more. These registers **MUST** hold 160-bit values and **MUST** be held in TCPA-shielded locations. Further, these registers **MUST** be non-volatile (values are maintained during the power-off state). A TPM implementation need not provide the same number of DIRs as PCRs.

## 6.3 Operations Supporting Integrity Collection and Reporting

### 6.3.1 TPM\_Extend

The TPM\_Extend operation MUST be the only operation that can modify PCR contents (beyond internal POST code and register initialization, which also happens only during POST).

#### IDL Definition

```
TCPA_RESULT TPM_Extend(
    [in] TCPA_PCRINDEX Pcrnum,
    [in] TCPA_DIGEST InDigest,
    [out] TCPA_PCRVALUE* OutDigest);
```

#### Type

TCPA protected capability.

#### Parameters

Type	Name	Description
TCPA_PCRINDEX	Pcrnum	Index of the PCR to be modified
TCPA_DIGEST	InDigest	Any 160-bit value representing the event to be recorded
TCPA_PCRVALUE*	OutDigest	Pointer to a DIGEST-sized memory location that is updated by the TPM_Extend operation to be the contents of the named PCR when internal processing is complete. If this parameter is NULL, no value is returned. If the TPM is disabled, NULL is returned.

#### Actions

The TPM\_Extend operation performs a TCPA-defined one-way operation on the contents of the named PCR. The operation is computationally unfeasible to reverse. TPM\_Extend MUST form an internal data structure consisting of the current value of PCR<sub>index</sub> concatenated with the event parameter in a TCPA-shielded location (to form a 320-bit data structure, with the current PCR contents first and the event second). The TPM MUST then calculate the SHA-1 hash of the composite structure and MUST store the resulting value back in PCR<sub>index</sub>.

Here is this operation represented in pseudocode:

```
TCPA_PCRVALUE = SHA-1 (cat(TCPA_PCRVALUE , Event))
```

The SHA-1 operation is defined in section 10, "Conformance Criteria."

The TPM\_Extend operation will succeed and its internal actions will be performed even if the TCPA\_PERSISTENT\_FLAG disable or the TCPA\_VOLATILE\_FLAG deactivated is TRUE. However, if the TPM is disabled or deactivated, NULL is returned for PcrFinal, and all operations that attempt to read this value will fail with TCPA\_DISABLED.

The TPM\_Extend operation operates normally if the TPM is not yet initialized.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_BADINDEX	PCR index does not exist
TCPA_FAIL	A critical internal error occurred



### 6.3.2 TPM\_PcrRead

The TPM\_PcrRead operation provides non-cryptographic reporting of the contents of a named PCR.

#### IDL Definition

```
TCPA_RESULT TPM_PcrRead(  
    [in] TCPA_PCRINDEX Pcrnum,  
    [out] TCPA_PCRVALUE* OutDigest);
```

#### Type

TCPA protected capability

#### Parameters

Type	Name	Description
TCPA_PCRINDEX	Pcrnum	Index of the PCR to be read
TCPA_PCRVALUE *	OutDigest	The current contents of the named PCR

#### Actions

The TPM\_PcrRead operation returns the current contents of the named register and its flags to the caller.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_BADINDEX	PCR index does not exist
TCPA_BAD_PARAMETER	One or more parameters is bad
TCPA_FAIL	POST failed, or another critical error occurred
TCPA_DISABLED	The TPM is disabled

### 6.3.3 TPM\_Quote

The TPM\_Quote operation provides cryptographic reporting of PCR values. A loaded identity key is required for operation. TPM\_Quote uses an identity key to sign a statement that names the current value of a chosen PCR and externally supplied data (which may be a nonce supplied by a Challenger).

TPM\_CERTIFYKEY and TPM\_Quote are the only operations that use TPM identity keys apart from those operations used to acquire identities.

#### IDL Definition

```
TCPA_RESULT TPM_Quote(
    [in, out] TCPA_AUTH* Auth,
    [AUTH, in] UINT32 SigBlobMaxSize,
    [AUTH, in] TCPA_PCR_SELECTION targetPCR,
    [AUTH, in] TCPA_KEY_SLOT Key1,
    [AUTH, in] TCPA_DIGEST ExternalData,
    [AUTH, in, out] UINT32* SigBlobSize,
    [AUTH, out] TCPA_PCR_COMPOSITE* PcrData
    [AUTH, out, size_is(*SigBlobSize), out] BYTE* SigBlob);
```

#### Type

TCPA protected capability; user must provide authorization to use the key indicated by the key1 parameter.

#### Parameters

Type	Name	Description
TCPA_AUTH	Auth	Authorization data for the identity key used for the signing operation.
UINT32	SigBlobMaxSize	Maximum permissible size of the signature blob
TCPA_PCR_SELECTION	targetPCR	This SHALL be the indication of which PCR registers are active in this quote operation
TCPA_KEY_SLOT	Key1	This SHALL be the slot identifier of the key to provide the quote
TCPA_DIGEST	ExternalData	160 bits of externally supplied data (typically a nonce provided by a server to prevent replay-attacks)
UINT32*	SigBlobSize	Set to the size of the returned signature blob
TCPA_PCR_COMPOSITE	pcrData	This SHALL be the digest values of all the active PCR registers selected for the quote operation
BYTE*	SignatureBlob	Pointer to memory that is to receive the signed data blob

#### Actions

The TPM MUST validate the authorization to use the key pointed to by key1.

The TPM MUST check that the targetPCR parameter is a consistent TCPA\_PCR\_SELECTION structure. If not, the TPM MUST return the error code TCPA\_BADINDEX.

If the **targetPCRSize** parameter value is **0x00**, the TPM MUST return the error code **TCPA\_BADINDEX**.

If the **targetPCRSize** parameter value is not **0x00**, the TPM\_Quote operation SHALL:

1. Assemble a **TCPA\_PCR\_COMPOSITE** data structure in a TPM-shielded location. The PCR indices in the **TCPA\_PCR\_COMPOSITE** structure SHALL be the same as those in the **targetPCR** parameter. This **TCPA\_PCR\_COMPOSITE** data structure SHALL be returned by the call.
2. Create a **TCPA\_COMPOSITE\_HASH** structure as described in section 10.4.5, using the **TCPA\_PCR\_COMPOSITE** structure as an input.
3. Incorporate the **TCPA\_COMPOSITE\_HASH**, information about the type of operation (**TPM\_QUOTE**), version information, and the **ExternalData** parameter into a **TCPA\_QUOTE\_INFO** structure.
4. Sign the **TCPA\_QUOTE\_INFO** structure, using SHA-1 for hashing and the **Key1** parameter as the encryption key.
5. Return the resulting signature value in **SignatureBlob**.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_BADINDEX	PCR index does not exist.
TCPA_INVALID_HANDLE	The key handle does not refer to an active identity key handle.
TCPA_BAD_PARAMETER	One or more parameters are invalid.
TCPA_FAIL	An internal error occurred, or POST or a previous self-test failed.
TCPA_DISABLED	The TPM is disabled.

### 6.3.4 TSS\_LogExtendEvent

***Start of informative comment:***

TSS\_LogExtendEvent can be used to provide a uniform way of logging the supporting data that is sometimes needed to interpret PCR values and composite PCR values. The events logged can have supporting validation certificates, or they may be other data structures. TCPA defines certain event-type information (for instance, validation certificates). Other application-specific types may be added using the naming convention described.

All pre-OS components that call TPM\_Extend should also call TSS\_LogExtendEvent for each TPM\_Extend operation. One of the parameters to TSS\_LogExtendEvent is the actual digest-sized event logged to TPM\_Extend. The other parameters to LogExtendEvent describe the PCR to which the event relates and the certificates (or other data) that aid in interpreting the composite PCR values.

Conceptually, the TSS will maintain an array of events, with the array format defined later. The TSS\_LogExtendEvent operation adds a new event to the end of the array associated with the named PCR. The TSS is free to reallocate event-log storage as it sees fit. The TSS also is free to maintain additional data structures that permit fast random access to events.

Logged information is retrieved using the TSS\_GetExtendEvent call, where events are accessed by index, and TSS\_GetExtendEventLog, which returns a pointer to a data structure that describes the entire log. The logs maintained by TSS\_LogExtendEvent need not be held in TCPA-shielded locations, and the logging and retrieval operations need not be TCPA-protected capabilities. This is because servers or other software can detect tampering with the log.

Upper-level software that uses the TPM need not use the TSS-provided TPM\_Extend log. Any OS-specific logging mechanism may be used. However, it is essential to transfer information in a uniform way between the pre-OS environment and the OS itself. Its use *is* required in the PC boot case for this purpose.

***End of informative comment.***

#### IDL Definition

```
TCPA_RESULT TSS_LogExtendEvent(
    [in] TCPA_PCRINDEX Pcr,
    [in] TCPA_PCRVALUE PcrValue,
    [in] UINT32 EventType,
    [in] UINT32 EventSize,
    [in, size_is(EventSize)] BYTE* Event,
    [out] UINT32* EventNumber);
```

#### Type

TSS function

#### Parameters

Type	Name	Description
TCPA_PCRINDEX	Pcr	PCR with which the event should be associated
TCPA_PCRVALUE	PcrValue	Parameter passed to the corresponding TPM_Extend operation
UINT32	EventType	Type of event. TCPA defines certain events and reserves others. The EventType parameter specifies the form of the supporting event information to make interpretation easier.

UINT32	EventSize	Size of the data structure containing the supporting information in bytes
BYTE*	Event	Pointer to an opaque data structure that provides the supporting information for an event
UINT32*	EventNumber	The number of the event just logged. The TSS numbers events <i>for each PCR</i> monotonically from 0 (i.e., events associated with each PCR are separately numbered from 0).

### Actions

The TSS\_LogExtendEvent operation MUST add supporting information for the named TPM\_Extend event to the end of TSS event log. The TSS MUST maintain an array of event-supporting data with events identified by the register to which they belong and the order in which the events occurred. The log need not be in a TCPA-shielded location, and the TSS\_LogExtendEvent action need not be a TCPA-protected capability. The TSS MUST NOT impose arbitrary size limitations on the size of the event log. The event log size should be limited by physical memory, memory accessible in the given operating mode, or memory allocated to the log by system firmware or other software.

It is anticipated that upper-level software will make a copy of needed event data and will dispose of the logs once copies are made. In all cases, ExtendValue should be the actual digest-sized event passed to TPM\_Extend.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_BADINDEX	PCR index does not exist
TCPA_BAD_PARAMETER	*Action is not readable
TCPA_RESOURCES	The event log is full

### 6.3.5 TSS\_GetExtendEvent

**Start of informative comment:**

TSS\_GetExtendEvent is used to retrieve events logged with TSS\_LogExtendEvent. TSS\_GetExtendEvent need not be a TCPA-protected capability, and the log events retrieved need not be in TCPA-shielded locations. TSS\_GetExtendEvent returns the event type reported to TSS\_LogExtendEvent, the DIGEST-sized event passed to TPM\_Extend, the opaque data blob provided as supporting information for the event, and its length.

TSS\_GetExtendEvent is not a TCPA-protected capability and does not access shielded data; hence, it cannot be protected against unauthorized access by the procedures available in this specification. However, TSS implementors may choose to provide their own restrictions against unauthorized access.

**End of informative comment.**

#### IDL Definition

```
TCPA_RESULT TSS_GetExtendEvent(
    [in] TCPA_PCRINDEX Pcr,
    [in] UINT32 EventNumber,
    [in] UINT32 EventMaxSize,
    [in, out] UINT32* EventSize,
    [out] UINT32* EventType,
    [out] TCPA_PCRVALUE* PcrValue,
    [out, size_is(*EventSize)] BYTE* Event);
```

#### Type

TSS function

#### Parameters

Type	Name	Description
TCPA_PCRINDEX	Pcr	PCR for which the event is being queried
UINT32	EventNumber	Index of event required. Events are numbered from 0 to 1 – the number of events logged on the named PCR
UINT32	EventMaxSize	Maximum acceptable size of the event data to be returned in bytes. If this parameter is zero, no actual data will be written into *Data, but the *Size parameter will be set to the size of the buffer required.
UINT32*	EventSize	Actual size of the event data returned in bytes, or size of the data buffer required
UINT32	EventType	The type of the event
TCPA_PCRVALUE*	PcrValue	Event parameter passed to TPM_Extend
BYTE*	Event	Pointer to a memory location that will be filled with the opaque binary data describing the event

#### Actions

The TSS\_GetExtendEvent operation retrieves events previously logged using TSS\_LogExtendEvent. The format of the data returned is identical to that previously logged. This operation interface retrieves log

entries by index. On TSS initialization (or following a TSS\_DisposeEventLog call), the event log for each PCR is empty. The first event logged to a register is numbered 0, the next is numbered 1, and so on. Attempts to receive log items beyond the end of the log return an error.

Note that the event log is required to be accessible in the form of an array (whose properties are defined in section 6.3.6). TSS implementation MAY choose to provide supplemental data structures to make random array access through TSS\_GetExtendEvent more efficient.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_BADINDEX	PCR index does not exist
TCPA_BADEVENT	The numbered event does not exist
TCPA_BAD_PARAMETER	One or more parameters are invalid
TCPA_BUFSIZE	The size specified in MaxSize is not large enough to hold the event data structure. If this error is returned, *Size is still set to the buffer size required.

### 6.3.6 TSS\_GetExtendEventLog

***Start of informative comment:***

TSS\_GetExtendEventLog returns a selected event from the log of all events since the TPM was initialized or since TSS\_DisposeEventLog (defined next) was called. The data structure returned is an array of TCPA\_PCR\_EVENT data structures. The array elements are of variable size, and the TCPA\_PCR\_EVENT structure defines the size of the current event and the register with which it is associated. This data structure is not required to be thread-safe, so upper-level software should ensure that it is not modified during parsing. The array terminator is a defined sentinel. If the event log is kept in a TCPA-shielded location, then a copy must be made in an unprotected area that can be traversed by non-TPM protected calling code.

TSS\_GetExtendEventLog is not a TCPA-protected capability and does not access shielded data; hence, it cannot be protected against unauthorized access by the procedures available in this specification. However, TSS implementors may choose to provide their own restrictions against unauthorized access.

***End of informative comment.***

#### IDL Definition

```
TCPA_RESULT TSS_GetExtendEventLog(
    [out] UINT32* Log);
```

#### Type

TSS function.

#### Parameters

Type	Name	Description
UINT32*	Log	The operation sets this pointer to point to the head of the event log list data structures.

Events are variably sized. The size of each event, including that of the Length and PCRIndex parameters, is specified in the TCPA\_PCR\_EVENT.Length parameter. The Event variable-sized array is the event data itself, and the PCRIndex is the register to which the event relates.

The whole event log is returned as a pointer to an array of these variably sized TCPA\_PCR\_EVENT structures. Individual TCPA\_PCR\_EVENT items are BYTE-aligned. The event log is terminated by a TCPA\_PCR\_EVENT element in which the TCPA\_PCR\_EVENT.Index is zero. The head of the array is returned using the TSS\_GetExtendEvent call.

#### Actions

This command returns to the caller the complete event log.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_BAD_PARAMETER	EventLogHead is NULL or memory is not writable.
TCPA_FAIL	A critical system error occurred.



### 6.3.7 TSS\_DisposeEventLog

**Start of informative comment:**

The TSS\_DisposeEventLog operation instructs the TSS to dispose of the event logs for all registers and optionally free memory in use. Calls to query the event log after TSS\_DisposeEventLog will indicate an empty log. Upper-level software may still log new events to the TSS, although this is unlikely to be useful, since partial logs are difficult to interpret. It is instead anticipated that upper-level software will maintain a copy of this pre-OS event log and dispose of the original.

**End of informative comment.****IDL Definition**

```
TCPA_RESULT TSS_DisposeEventLog();
```

**Type**

TSS function

**Parameters**

Type	Name	Description
None		

**Actions**

DISPOSE\_EVENT\_LOG empties all logs entries for all registers, and optionally frees memory associated with the data structures. It has no effect on the PCRs themselves.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_FAIL	A critical system error occurred.

### 6.3.8 TPM\_DirWriteAuth

***Start of informative comment:***

The TPM\_DirWriteAuth operation provides write access to the Data Integrity Registers. DIRs are non-volatile memory registers held in a TCPA-shielded location. Owner authentication is required to authorize this action. TCPA\_version 1.0 requires only one DIR. If the DIR named does not exist, the TPM\_DirRead operation returns TCPA\_BADINDEX.

***End of informative comment.***

**IDL Definition**

```
TCPA_RESULT TPM_DirWriteAuth(
    [in, out] TCPA_AUTH* OwnerAuth,
    [AUTH, in] TCPA_DIRINDEX DIRindex,
    [AUTH, in] TCPA_DIRVALUE NewContents);
```

**Type**

TCPA protected capability; the user must provide authorization from the TPM Owner to execute function.

**Parameters**

Type	Name	Description
TCPA_AUTH*	OwnerAuth	Owner-authentication data for an active session.
TCPA_DIRINDEX	DIRindex	Index of the DIR.
TCPA_DIRVALUE	NewContents	Value to be stored in the named DIR.

**Actions**

This is an Owner-authenticated action. In order to perform this action, software must have a valid Owner session. TPM\_DirWriteAuth allows the Owner, after authentication, to write a new value into the named DIR.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_BADINDEX	The DIR index does not exist.
TCPA_AUTHFAIL	Authentication failed.
TCPA_FAIL	An internal error occurred, or an earlier self-test failed.
TCPA_DISABLED	The TPM is disabled.

### 6.3.9 TPM\_DirRead

***Start of informative comment:***

The TPM\_DirRead operation provides read access to the DIRs. No authentication is required to perform this action because typically no cryptographically useful authorization data is available early in boot. TSS implementors may choose to provide other means of authorizing this action. TCPA\_version 1.0 requires only one DIR. If the DIR named does not exist, the TPM\_DirRead operation returns TCPA\_BADINDEX.

***End of informative comment.***

#### IDL Definition

```
TCPA_RESULT TPM_DirRead(
    [in] TCPA_DIRINDEX DIRindex,
    [in, out] TCPA_DIRVALUE* Contents);
```

#### Type

TCPA protected capability.

#### Parameters

Type	Name	Description
TCPA_DIRINDEX	DIRIndex	Index of the DIR.
TCPA_DIRVALUE*	Contents	Pointer to a memory location that will receive the contents.

#### Actions

TPM\_DirRead is a non-authenticated operation that returns the current contents of the named DIR.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_BADINDEX	The DIR does not exist.
TCPA_FAIL	An internal error occurred, or an earlier self-test failed.

## 7. Protected Storage

### ***Start of informative comment:***

This section introduces the processes by which a TPM may act as the portal to confidential data stored on arbitrary storage media.

A TPM is required to protect the keys that represent TPM identities, and keys that are released only when the computing environment of the associated platform has a particular state. Given this capability, it is a natural extension to enable a TPM to protect arbitrary data and arbitrary keys. Unfortunately, this approach requires a potentially unbounded amount of storage within a TPM. The TCPA specification therefore includes capabilities that enable a TPM to act as a portal to potentially unbounded amounts of confidential data outside the TPM.

Storing data outside the TPM has the additional advantages of enabling easier migration of confidential data from one platform to another and enabling recovery of confidential data in the event of platform failure. These protected-storage capabilities are designed to enable the TPM to operate as a slave device so as to avoid the cost complexity associated with a master device in a computing platform. These capabilities also are designed to avoid the need for the TPM to manage the confidential data that is stored outside the TPM. These design goals impose constraints on the nature of the protected-storage capabilities.

The TCPA solution uses the TPM to generate “blobs” of secret data. Unspecified capabilities outside the Subsystem manage protected storage and issue certificates or other indications about the purpose and usefulness of data/keys held in blobs. Those unspecified capabilities issue commands to the TPM that cause it to create blobs of data and to use and return the contents of such blobs. This unspecified functionality is the manager of protected storage and uses the TPM as a specialized co-processor. The protected-storage commands are chosen to prevent subversion of the data in protected storage. Hence a rogue management function can disrupt protected storage but cannot subvert it.

A stored secret could be any of the following:

- Arbitrary data or a key. If a secret is arbitrary data, it can be exported from the TPM, and the TPM will not perform operations using that data. If the secret is a key, it is available for use within the TPM, and will never be exported from the TPM.
- An encryption (storage) key or a signing key. If a key is for encryption, it must not be used for signing, and visa versa. Encryption keys are used only to provide confidentiality for blobs. Signature keys are used for signing arbitrary data submitted by the entity authorized to use that key.
- The signature key of a TPM identity. Such a signature key will be used only for special signing operations.

A stored secret has the following attributes:

- It may be capable of migration to another platform or it may be non-migratable. Keys that are migratable cannot be considered unique to a particular platform. Non-migratable keys can be considered to be unique to a particular platform.
- It may be generated inside the TPM or externally loaded. Externally loaded keys cannot be stored as non-migratable keys, for obvious reasons.
- It may be bound to the TPM or bound to a sequence of integrity metrics. At times, data or a key is required to be bound to a particular platform. At other times, it is required to be bound to a particular computing environment within a platform.
- It may have access control. A secret may be open to all processes on a platform or it may not, with varying degrees of control in between.

Some of these attributes are partitioned as separate commands, while others are partitioned as flags within commands. All the commands cause the TPM to create a secret blob and return it to the caller. The inverse commands cause the TPM to import a blob. Sometimes the TPM will then return the contents of

the blob (data) to the caller, and sometimes the TPM loads the contents of the blob (a key) for use within the TPM.

In all cases, the TPM must already contain the key that will be used to either encrypt or decrypt the blob. This naturally leads to a tree of blobs, where intermediate nodes contain encryption (storage) keys that are used to encrypt/decrypt child nodes. The root of the tree is the “Storage Root Key” (SRK) which is generated inside the TPM and is non-migratable. Only leaf nodes can contain signing keys, because a TPM will refuse to use a signing key to encrypt/decrypt child nodes. A TPM also will refuse to use a migratable node as the parent of a non-migratable node. (This enables migration of the supposedly non-migratable node.) On the other hand, a non-migratable node could be the parent of a migratable node, with no ill effects.

The commands executed by the TPM are as follows:

- **TSS\_Bind:** External data is encrypted under a parent key. (TPM\_UnBind decrypts the blob using the parent key and exports the data from the TPM.)
- **TPM\_Seal:** External data is concatenated with a value of integrity metric sequence and encrypted under a parent key. (TPM\_Unseal decrypts the blob using the parent key and exports the plaintext data if the current integrity metric sequence inside the TPM matches the value of integrity metric sequence inside the blob.)
- **TSS\_WrapKey:** An externally generated key is encrypted under a parent key. (TPM\_LoadKey decrypts the target blob using the parent key and loads the target key inside the TPM, for use by the TPM.)
- **TSS\_WrapKeyToPcr:** An externally generated key is concatenated with a value of integrity metric sequence and encrypted under a parent key. (TPM\_LoadKey decrypts the target blob using the parent key and loads the target key inside the TPM, for use by the TPM, if the current integrity metric sequence inside the TPM matches the value of integrity metric sequence inside the blob.)
- **TPM\_CreateWrapKey:** A key is generated inside the TPM and then encrypted under a parent key. (TPM\_LoadKey decrypts the target blob using the parent key and loads the target key inside the TPM, for use by the TPM.)
- **TPM\_CreateWrapKeyToPcr:** A key is generated inside the TPM, concatenated with a value of integrity metric sequence, and encrypted under a parent key. (TPM\_LoadKey decrypts the target blob using the parent key and loads the target key inside the TPM, for use by the TPM, if the current integrity metric sequence inside the TPM matches the value of integrity metric sequence inside the blob.)

When a blob is loaded into a TPM, the TPM distinguishes between a data-bearing blob and a key-bearing blob by inspecting the data structure inside the blob. Data-bearing blobs are constructed according to PKCS #1. Key-bearing blobs are constructed using a TCPA-defined format. Each blob containing a key includes the field KeyUsage, which indicates whether the key is to be used for encryption (storage) or signing.

Command	Usage with keys	Comment
TSS_Bind	N/A	No key
TPM_Seal	N/A	No key
TSS_WrapKey	Migratable, encrypt or sign	Externally loaded
TSS_WrapKeyToPcr	Migratable, encrypt or sign	Externally loaded
TPM_CreateWrapKey	Any	
TPM_CreateWrapKeyToPcr	Any	

TCPA-protected storage uses asymmetric cryptography exclusively. One reason is that asymmetric crypto is already required to support TPM identities, but asymmetric crypto is not specifically necessary for any function. Another reason is that (in many, but not all, cases) operations to construct blobs can be performed outside the TPM; only the recovery of information from blobs (using the private key) must be

done inside a TPM. This is possible because it is frequently true that all the necessary data to construct a blob (including the public key) is available outside the TPM. One notable exception is the TPM\_Seal command, which must be performed inside a TPM because it requires reliable access to the Platform Configuration Registers. Using asymmetric crypto for protected storage therefore reduces the complexity of a TPM.

Some other important characteristics of “protected storage” are

- Whenever a blob is created, the TPM includes random data to guard against plaintext attacks.
- Whenever a CreateXX command creates a new key within the TPM, the blob that is produced contains the private (signature) key and the TPM also exports the corresponding public (identity) key as plaintext.
- Whenever a WrapXX command adds a new key into the TPM, only the private key (and its RSA modulus) must be presented.
- Whenever the TPM\_LoadKey command is asserted, the TPM imports a secret blob containing the private (signature) key and the TPM also imports the corresponding public (identity) key as plaintext. Active RSA keys inside the TPM are referenced by slot number when loaded into the TPM. To minimize key management burden inside the TPM, it is assumed “key slot” management is performed outside the TPM.
- The integrity of the data from the TPM\_UnBind command is not checked by the TPM. Hence applications should use an “out of band” mechanism for verifying data integrity, if such verification is necessary.

Each secret blob contains a field of 20 bytes that may be used for authorization data. For convenience, the authorization field is the same size as the output of the SHA-1 hash algorithm. The authorization field is merely stored inside a blob, and the protected-storage capabilities do not themselves interpret the field.

The AuthorizationDataUsage field determines when authorization is required.

The integrity of data or keys recovered from blobs is ensured by an implicit, rather than explicit, mechanism. Ordinarily, an integrity check is provided by appending a checksum to original plaintext data. After decryption, the checksum is recomputed and compared with the checksum in the recovered data. Such a checksum needs to be at least 16 bytes long so as to have the necessary statistical properties. In the case of recovered blobs, the first 20 bytes of authorization data are sufficient to determine with high probability that data has been successfully decrypted without error. If the decryption fails, or the encrypted data contains errors, it is unlikely that the authorization data in the recovered blob will match the submitted authorization data.

The TPM also can be commanded to provide evidence that a particular public key is associated with a non-migratable private key (which was generated by the TPM and has never been released outside the TPM). This is the TPM\_CERTIFYKEY operation. It enables a third party to use a public key to encrypt data that can be recovered only using a protected-storage command. It also enables a third party to have confidence that a signature key has been generated by the TPM and has never been released outside the TPM.

Migratory data may be copied to an arbitrary number of platforms, using the “migration” commands provided. Non-migratory data may be moved to another platform only with the cooperation of a third party (the manufacturer of the platform, or his representative), using the “maintenance” commands provided.

***End of informative comment.***

## 7.1 Introduction

### 7.1.1 Characteristics

***Start of informative comment:***

This section specifies how to use the TPM to provide secure storage for an unlimited number of private keys or other data. Basically, this is done through the RSA key technology built into the TPM to encrypt data and keys with a public key to which the TPM has access to its corresponding private key. The resulting encrypted file, which contains header information in addition to the data or key, is called a blob, and cannot be any bigger than key size used to encrypt it. The specification also shows how this is done, so that private keys generated on the TPM can be stored outside the TPM (encrypted) in a way that allows the TPM to use them later without ever exposing such keys in the clear outside the TPM.

Padding and speed requirements make the TPM a very inefficient and inappropriate vehicle to do any bulk encryption, but it can be used to securely store keys that would then be used by software to do bulk encryption. There are a number of usage modules that imply requirements on the function of the TPM, as follows:

- Signing with a private key by the TPM can be accomplished only by presentation of authorization data to the TPM that is associated with that private key. A private key generated by a third party can be linked to a specific TPM without exposing the private key to the Owner/User of the TPM, but only with the consent of the User of the TPM.
- It MUST be possible to prove a specific public key is associated with a private key known only to a TPM. It must be possible for the Owner of a key, with the cooperation of the Owner of the TPM to migrate a migratable key from one platform to another without giving up control of the key to the TPM Owner.
- It must not be possible for the Owner of a key, even with the cooperation of the Owner of the TPM to migrate a non-migratable key from one platform to another. Since a key may be wrapped outside the TPM, it is necessary that non-migratable keys always be generated inside the TPM. It must not be possible for the Owner of a non-migratable asymmetric key, even with cooperation of the Owner of the TPM, to decrypt the contents of an encrypted bundle encrypted with that non-migratable asymmetric key.
- If a TPM is compromised, it must not compromise all TPMs.
- To facilitate application level exchange of symmetric keys, the symmetric keys are stored using PKCS#1.

All this is generally accomplished as follows:

- Any data in protected storage is explicitly identified as migratable or non-migratable.
- Each TPM contains a SRK, generated by the TPM at the request of the Owner. Under that SRK are two trees: one dealing with migratable data and the other dealing with non-migratable data.
- The non-migratable tree is directly below the SRK. The migration tree is directly below a “migration root” key that is directly below the SRK. Each node in a tree provides confidentiality for the nodes immediately below it. Obviously, all intermediate nodes in the trees must be encryption keys. Nodes in the non-migratable tree must be generated by the TPM; otherwise, non-migratable nodes could be exposed.

Finally, some observations:

- In the migration tree, only leaf nodes should be available for signing. This is because a signature node (used outside the TPM for signing) should never be used for encryption and hence cannot be used to encrypt other nodes. Hence, it must be a leaf.
- Similarly, in a non-migration tree, only leaf-nodes should be available for signing. Since non-migratable nodes must not be migrated, they must never appear outside the TPM after being installed in the TPM.
- Any non-leaf node in the non-migratable tree must be generated within the TPM and never exposed outside the TPM. Any key (and hence every non-migratable key) generated in a TPM must be a genuine key.

- Any migratable key can be migrated by anyone that owns any of its migratable ancestors. As a result, in order to be sure that a migratable key cannot be migrated by anyone but the owner of that key, the owner can always create the migratable key and store it with a non-migratable storage key, thus guaranteeing the user has unique authority to authorize migration of that key.

***End of informative comment.***

### 7.1.2 Key Storage

The number of asymmetric keys that are storable via a TPM SHOULD be limited only by the volume of storage available to the platform.

The TPM SHALL ensure that the keys in all slots, other than slot 0, are volatile.

## 7.2 Mandatory Functions

***Start of informative comment:***

Every TSS MUST support these functions; some must be TPM, and all may be TPM. They are derived from three parameters:

1. Is the secret stored data or as a key?
2. Is the secret generated internally or externally?
3. Is the secret bound to just the platform or also to PCRs?

These parameters would ordinarily lead to eight functions, but because data is always assumed to be generated externally, they yield to just six functions, as follows:

1. Data, generated externally, bound to PCRs: TPM\_Seal command (TPM-protected capability). Inverse command is TPM\_Unseal.
2. Data, generated externally, bound to platform: TSS\_Bind command (TSS). Inverse command is TPM\_UnBind.
3. Key, generated internally, bound to PCRs: TPM\_CreateWrapKeyToPcr command (TPM-protected capability). Inverse command is TPM\_LoadKey.
4. Key, generated externally, bound to PCRs: TSS\_WrapKeyToPcr (TSS). Inverse command is TPM\_LoadKey.
5. Key, generated internally, bound to platform: TPM\_CreateWrapKey command (TPM-protected capability). Inverse command is TPM\_LoadKey.
6. Key, generated externally, bound to platform: TSS\_WrapKey command (TSS). Inverse command is TPM\_LoadKey.

***End of informative comment.***



### 7.2.1 TPM\_Seal

***Start of informative comment:***

The SEAL operation allows software to explicitly state the future “trusted” configuration that the platform must be in for the secret to be revealed. The SEAL operation also implicitly includes the relevant platform configuration (PCR-values) when the SEAL operation was performed.

If the UNSEAL operation succeeds, proof of the platform configuration that was in effect when the SEAL operation was performed is returned to the caller, as well as the secret data. This proof may, or may not, be of interest. If the SEALED secret is used to authenticate the platform to a third party, a caller is normally unconcerned about the state of the platform when the secret was SEALED, and the proof may be of no interest. On the other hand, if the SEALED secret is used to authenticate a third party to the platform, a caller is normally concerned about the state of the platform when the secret was SEALED. Then the proof is of interest.

For example, if SEAL is used to store a secret key for a future configuration (probably to prove that the platform is a particular platform that is in a particular configuration), the only requirement is that that key can be used only when the platform is in that future configuration. Then there is no interest in the platform configuration when the secret key was SEALED. An example of this case is when SEAL is used to store a network authentication key.

On the other hand, suppose an OS contains an encrypted database of users allowed to log on to the platform. The OS uses a SEALED blob to store the encryption key for the user-database. However, the nature of SEAL is that *any* SW stack can SEAL a blob for any other software stack. Hence the OS can be attacked by a second OS replacing both the SEALED-blob encryption key, *and* the user database itself, allowing untrusted parties access to the services of the OS. To thwart such attacks, SEALED blobs include the *past* SW configuration. Hence, if the OS is concerned about such attacks, it may check to see whether the past configuration is one that is known to be trusted.

***End of informative comment.***

#### IDL Definition

```
TCPA_RESULT TPM_Seal(
    [in, out] TCPA_AUTH* pubAuth,
    [AUTH, in] UINT32 futurePCRSize,
    [AUTH, in] UINT32 blobSize,
    [AUTH, in] UINT32 SealedMaxSize,
    [AUTH, in] BOOL CurrentStateOut,
    [AUTH, in] TCPA_ENCAUTH Secret,
    [AUTH, in] TCPA_KEY_SLOT parentSlot,
    [AUTH, in] TCPA_COMPOSITE_HASH targetPCRHash,
    [AUTH, in, size_is(futurePCRSize)] BYTE* targetPCR,
    [AUTH, in, size_is(blobSize)] BYTE* blob,
    [AUTH, in, out] UINT32* SealedSize,
    [AUTH, in, out] UINT32* currentPCRSize,
    [AUTH, out, size_is(*SealedSize), out] BYTE* SealedBlob,
    [AUTH, out, size_is(*currentPCRSize), out] BYTE* currentPCR);
```

#### Type

TPM function; user must provide authorization to use the key pointed to by keySlot.

#### Parameters

Type	Name	Description
TCPA_AUTH*	PubAuth	This SHALL be the authorization session that authorizes the use of key pointed to by keySlot.

		authorizes the use of key pointed to by keySlot. The session type MUST be OS-AP.
UINT32	futurePCRSize	This SHALL be the size of the futurePCR parameter
UINT32	blobSize	This SHALL be the size of the object parameter
UINT32	SealedMaxSize	The maximum size of the output area
BOOL	CurrentStateOut	If set to FALSE, the TPM SHALL not return the current PCR values.
TCPA_ENCAUTH	Secret	The encrypted authorization data for the sealed data. The encryption key is the shared secret from the OS-AP protocol being an XOR of the data.
TCPA_KEY_SLOT	parentSlot	This SHALL be the public key that is the parent of the sealed data
TCPA_COMPOSITE_HASH	TargetPCRHash	This SHALL be the composite digest of the PCR indexes and values to which parameter blob is to be sealed. This must have been constructed according to the algorithm described in 10.4.5 using the target PCR values.
BYTE*	targetPCR	This SHALL be a TPCA_PCR_SELECTION structure containing the list of the indexes of the PCRs to which the blob parameter is to be sealed.
BYTE*	Blob	This SHALL be the data to be sealed to the platform and any specified PCRs
UINT32*	SealedSize	The used size of the output area for SealedBlob
UINT32*	currentPCRSize	The used size of the output area for currentPCR
BYTE*	SealedBlob	Encrypted, integrity-protected data object that is the result of the TPM_Seal operation.
BYTE*	CurrentPCR	This SHALL be the concatenated TPCA_PCR_COMPOSITE structure as computed by the TPM with the current PCR values

## Actions

TPM\_Seal is used to encrypt private objects that can only be decrypted using TPM\_Unseal.

The TPM\_Seal command MUST use the RSAES\_OAEP protocol from PKCS#1 version 2.0 to perform the encryption.

The TPM\_Seal command MUST fill in a TPM\_SEALED\_DATA structure and then encrypt the structure. The encryption key for the operation is the key pointed to by parentSlot parameter.

The TPM MUST check that the targetPCR parameter is a consistent TPCA\_PCR\_SELECTION structure. If not, the TPM MUST return the error code TPCA\_BADINDEX.

**If the targetPCRSize parameter value is not equal to 0x00**, the TPM will compute a TPCA\_COMPOSITE\_HASH value, using the targetPCR TPCA\_PCR\_SELECTION structure, to fill the

TPM\_SEALED\_DATA.digestAtCreation member variable. The TPM MUST compute this TCPA\_COMPOSITE\_HASH value using the targetPCR parameter as the input to the algorithm described in 10.4.5. The TPM MUST set the TPM\_SEALED\_DATA.IsSealedToPCR value to TRUE.

**If the targetPCRSize parameter value is 0x00**, then the blob parameter is not bound to any particular PCR values. The TPM MUST set the TPM\_SEALED\_DATA.IsSealedToPCR value to FALSE.

**If the CurrentStateOut parameter is set to TRUE**, then the currentPCR parameter MUST be the TCPA\_PCR\_COMPOSITE structure that was generated by the TPM when creating the TPM\_SEALED\_DATA.digestAtCreation.

**If the CurrentStateOut parameter is set to FALSE**, then the TPM MUST not return the current values of the PCRs in targetPCR.pcrIndex. The TPM MUST set CurrentPCRSize to 0. Note that in this case, the TPM still computes the TPM\_SEALED\_DATA.digestAtCreation parameter.

The TPM SHALL return TCPA\_FAIL CurrentStateOut is TRUE and targetPCRSize equals 0.

While the caller MUST provide authorization data, there is no requirement on the authorization data itself. If the caller wishes to use authorization data like nulls or other well-known values the TPM MUST NOT check for these conditions.

Manufacturers MUST ensure that TPM\_Sealed blobs are distinguishable by the TPM from other encrypted data blobs by using the TPM\_SEALED\_DATA structure.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_BADINDEX	PCR index does not exist.
TCPA_FAIL	An internal error occurred, or a previous self-test failed.
TCPA_AUTHFAIL	Authorization data is incorrect.
TCPA_DISABLED	The TPM is disabled.

## 7.2.2 TPM\_Unseal

### **Start of informative comment:**

The TPM\_Unseal operation will reveal TPM\_Sealed data only if it was encrypted on this platform and the current configuration (as defined by the named PCR contents) is the one named as qualified to decrypt it. Internally, TPM\_Unseal accepts a data blob generated by a TPM\_Seal operation. TPM\_Unseal decrypts the structure internally, checks the integrity of the resulting data, *and* checks that the PCR named has the value named during TPM\_Seal. Additionally, the caller must supply appropriate authorization data for blob and for the key that was used to seal that data.

If the integrity, platform configuration and authorization checks succeed, the sealed data and a proof of the configuration of the platform at the time when the data was stored are returned to the caller; otherwise, an error is generated.

### **End of informative comment.**

### **IDL Definition**

```
TCPA_RESULT TPM_Unseal(
    [in, out] TPCA_AUTH* ParentAuth,
    [in, out] TPCA_AUTH* BlobAuth,
    [AUTH, in] UINT32 BlobSize,
    [AUTH, in] UINT32 MaxSecretSize,
    [AUTH, in] UINT32 pcrListSize,
    [AUTH, in] TPCA_KEY_SLOT parentKeySlot,
    [AUTH, in, size_is(BlobSize)] BYTE* Blob,
    [AUTH, in, size_is(pcrListSize)] BYTE* pcrList,
    [AUTH, in, out] UINT32* SecretSize,
    [AUTH, out] TPCA_COMPOSITE_HASH* OldPcrHash,
    [AUTH, out, size_is(*SecretSize)] BYTE* Secret);
```

### **Type**

TPM protected capability; the user must provide authorizations to use the parent key pointed to by parentKeySlot.

### **Parameters**

Type	Name	Description
TCPA_AUTH*	ParentAuth	Authorization data to use the key pointed to by the parentKeySlot parameter.
TCPA_AUTH*	BlobAuth	Authorization data to reveal the data encrypted in the blob parameter.
UINT32	BlobSize	Size of blob
UINT32	MaxSecretSize	Maximum size of the output secret
UINT32	pcrListSize	This SHALL be the size of the pcrList parameter
TCPA_KEY_SLOT	ParentKeySlot	This SHALL be the index of the TPM Slot where the key to be used for decryption of the blob is to be found.
BYTE*	Blob	Encrypted data blob generated by a TPM_Seal operation.

BYTE*	pcrList	This SHALL be a TPCA_PCR_SELECTION structure containing the list of the indexes of the PCRs to which the blob parameter is to be sealed.
UINT32*	SecretSize	The size of the outputted secret
TCPA_PCRVALUE*	OldPcrHash	This SHALL be the composite hash that was stored in the decrypted blob's digestAtCreation parameter.
BYTE*	Secret	Filled with the decrypted sealed data (if the operation succeeds).

### Actions

The TPM\_Unseal MUST decrypt the data blob into a TPCA-shielded location using the private part of the key pointed to by parentKeySlot. The decryption operation requires valid authorization data to use the that private key. If the authorization data is improper, the TPM MUST return the error TPCA\_AUTHFAIL.

The TPM MUST then check the integrity of the decrypted data blob. The integrity check establishes that the decrypted blob is a consistent TPM\_SEALED\_DATA structure created with by a TPM\_Seal operation on the same TPM that is attempting the TPM\_Unseal *and* that the data blob has not been modified. The TPM MUST check that the tpmProof parameter in the decrypted blob matches the TPM's own TPCA\_PERSISTENT\_FLAGS.tpmProof. If the decrypted blob fails the integrity checks, then the TPM\_Unseal operation MUST return the error TPCA\_NOTSEALEDBLOB.

**If the decrypted blob's IsSealedToPCR parameter value is TRUE**, then the TPM MUST ensure that the PCRs to which the blob was sealed are the same as the PCRs' values that exist at the time of TPM\_Unseal. To do this, the TPM will compute a composite hash using the pcrList parameter as the input to the composite hashing algorithm (See 10.4.5).

If the resulting composite hash matches the decrypted blob's digestAtUnseal parameter the TPM MUST return the data parameter of the decrypted blob as the TPM\_Unseal Secret output parameter, and the decrypted blob's digestAtCreation as the TPM\_Unseal OldPcrHash output parameter. If the composite hashes do no match, the TPM MUST return TPCA\_WRONGPCRVAL.

If the pcrList.pcrCount parameter is 0, the TPM MUST not unseal the data, and simply return the TPCA\_NO\_PCR\_INFO error return status.

The TPM MUST check that the pcrList parameter is a consistent TPCA\_PCR\_SELECTION structure. If not, the TPM MUST return the error code TPCA\_BADINDEX.

**If the decrypted blob's IsSealedToPCR parameter value is FALSE**, then the TPM does not need to check PCR configuration. The TPM MUST return the data parameter of the decrypted blob as the TPM\_Unseal Secret output parameter, and the decrypted blob's digestAtCreation as the TPM\_Unseal OldPcrHash output parameter (Although this OldPcrHash parameter has no meaning in this case).

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_BADINDEX	PCR index does not exist.
TCPA_NOTSEALEDBLOB	The encrypted blob is not a valid TPM_SeaLED blob created by this TPM.
TCPA_INVALID_HANDLE	The key handle does not exist or is not active.
TCPA_FAIL	An internal error occurred, or a previous self-test failed.
TCPA_WRONGPCRVAL	The named PCR value does not match the current PCR value.

TCPA_NO_PCR_INFO	The list of PCR indices to which the data is sealed has not been provided
TCPA_AUTHFAIL	The authorization data is improper.
TCPA_DISABLED	The TPM is disabled.

### 7.2.3 TSS\_Bind

**Start of informative comment:**

The TSS\_Bind command can either generate data and create a secure storage bundle for that data or merely create a secure storage bundle for data passed to it. The Generate provision may be used to create a random key for usage externally by a bulk encryption engine or by the TPM for functions other than those required by this specification.

**End of informative comment.**

#### IDL Definition

```
TCPA_RESULT TSS_Bind(
    [in] BOOL Generate,
    [in] UINT32 BlobSize,
    [in] UINT32 MaxOutBlobSize,
    [in] TCPA_PUBKEY PubKey,
    [in, size_is(BlobSize)] BYTE* Blob,
    [in,out] UINT32* OutBlobSize,
    [out, size_is(*OutBlobSize)] BYTE* OutBlob);
```

#### Type

TSS function

#### Parameters

Type	Name	Description
BOOL	Generate	TRUE means that TSS_BIND both generates and binds data. FALSE means that TSS_BIND binds submitted data (and does not generate it)
UINT32	BlobSize	Size of the data being bound.
UINT32	MaxOutBlobSize	The maximum size of the output blob
TCPA_PUBKEY	PubKey	Public key used to Bind the data
BYTE*	Blob	Data being bound.
UINT32*	OutBlobSize	Size of output blob
BYTE*	OutBlob	This is the bound data

#### Actions

TSS\_Bind will take one of two actions depending on the Generate parameter.

The data size MUST be 44 bytes less than the modulus of the PubKey.

##### Generate = true

- Generate random data of the size specified by BlobSize
- Create PKCS#1 data format
- Encrypt data area using public key specified in PubKey

##### Generate = false

- Format Blob parameter into PKCS#1 format
- Encrypt the data using the public key specified in PubKey

Both actions return the encrypted blob in the OutBlob parameter.

A check SHOULD be made that the bound data will not look to the TPM like a wrapped key (in which case the TPM would not be able to TPM\_UnBind the data).

Return Value	Description
TCPA_SUCCESS	Success
TCPA_FAIL	Unknown error



## 7.2.4 TPM\_UnBind

### **Start of informative comment:**

TPM\_UnBind takes the data blob that is the result of a TSS\_Bind command and decrypts it for export to the User. The caller must authorize the use of the key that will decrypt the incoming blob.

### **End of informative comment.**

### **IDL Definition**

```

TCPA_RESULT TPM_UnBind(
    [in, out] TCPA_AUTH* PubAuth,
    [AUTH, in] UINT32 BlobSize,
    [AUTH, in] TCPA_KEY_SLOT keySlot,
    [AUTH, in] UINT32 MaxOutSize,
    [AUTH, in, size_is(BlobSize)] BYTE* Blob,
    [AUTH, in, out] UINT32* OutSize,
    [AUTH, size_is(*OutSize), out] BYTE* OutArea);

```

### **Type**

TCPA protected capability; the user must provide authorization to use the key specified in the pubKey parameter.

### **Parameters**

Type	Name	Description
TCPA_AUTH*	PubAuth	HMAC authorization for TPM to use the private key to decrypt the Blob.
UINT32	BlobSize	Size of input blob
TCPA_KEY_SLOT	keySlot	Slot containing private key corresponding the to public key used to bind the data in TSS_Bind
UINT32	MaxOutSize	Maximum allowed output size
BYTE*	Blob	Encrypted Blob to be decrypted.
UINT32*	Outsize	Length of output data
BYTE*	OutArea	The secret that was inside the Blob, now decrypted.

### **Actions**

The TPM SHALL perform the following:

- Validate the authorization to use the key pointed to by keySlot
- Decrypt the Blob using the key pointed to by keySlot
- Return the decrypted information in parameter OutArea

Return Value	Description
TCPA_SUCCESS	Success.
TCPA_AUTHFAIL	Authorization failed.
TCPA_FAIL	Unknown error

### 7.2.5 TPM\_CreateWrapKey

**Start of informative comment:**

The TPM\_CreateWrapKey command both generates and creates a secure storage bundle for asymmetric keys.

**End of informative comment.**

**IDL Definition**

```
TCPA_RESULT TPM_CreateWrapKey(
    [in, out] TCPA_AUTH* KeySlotAuth,
    [AUTH, in] TCPA_ENCAUTH DataUsageAuth,
    [AUTH, in] TCPA_ENCAUTH DataMigrationAuth,
    [AUTH, in] TCPA_KEY_SLOT KeySlot,
    [AUTH, in] UINT16 Type,
    [AUTH, in] TCPA_KEYUSAGE KeyUsage,
    [AUTH, in] TCPA_DATAUSAGE DataUsage,
    [AUTH, in, out] TCPA_KEY keyInfo;
```

**Type**

TCPA protected capability; user must provide authorization to use the key pointed to by keySlot.

**Parameters**

Type	Name	Description
TCPA_AUTH	KeySlotAuth	Authorization to use key in keySlot. Session type MUST be OSAP.
TCPA_ENCAUTH	DataUsageAuth	Encrypted authorization data to be used with the new key.
TCPA_ENCAUTH	DataMigrationAuth	Encrypted data used to authorize migration of the newly generated key.
TCPA_KEY_SLOT	KeySlot	Public key used to wrap the generated key
UINT16	Type	Used for nonstandard keys.
TCPA_KEYUSAGE	KeyUsage	Indicates the usage of the key
TCPA_DATAUSAGE	DataUsage	Indicates how the data will be used
TCPA_KEY	keyInfo	The input structure contains all parameters except pubkey and privkey (which are NULL), to specify the size and type of the new key. The output structure also contains pubkey (the public part of the new key) and privkey (the wrapped private key).

**Descriptions**

This command requires the encryption of two parameters. To create two XOR strings the caller takes the two nonces in use by the OSAP session and concatenates one nonce and the session shared secret and then hashes the result. The hash from the shared secret and the odd numbered hash (generated by the caller) encrypts the DataUsageAuth. The hash from the shared secret and the even numbered hash (generated by the TPM) encrypts the DataMigrationAuth.

**Actions**

The TPM SHALL do the following:

- Validate the authorization to use the key pointed to by KeySlot. Return TCPA\_BDAUTH on any error.
- If the key in KeySlot is migratable and requested key is non-migratable then return TCPA\_MIGRATEFAIL.
- Validate the key in KeySlot is a storage key
- Validate all other parameters
- Any error on above checks return TCPA\_BAD\_PARAMETER
- Create the two XOR patterns by using the session key and the nonces for this transaction
- Decrypt the DataUsageAuth and DataMigrationAuth parameters
- Generate asymmetric key according to parameters sent
- Create a TCPA\_KEY structure using the key pointed to by KeySlot as the key for any encryptions
- Return the TCPA\_KEY structure in the keyInfo parameter

Return Value	Description
TCPA_SUCCESS	Success
TCPA_FAIL	General error
TCPA_BAD_PARAMETER	One of the parameter was in error
TCPA_AUTHFAIL	The authorization to use keySlot failed

## 7.2.6 TPM\_CreateWrapKeyToPcr

### **Start of informative comment:**

The TPM\_CreateWrapKeyToPcr is similar to the TPM\_CreateWrapKey command except that TPM\_CreateWrapKeyToPcr locks the data blob to a PCR value as well as authorization data, and wraps only with a non-migratable key. This command generates *and* creates a secure storage bundle for asymmetric keys. .

### **End of informative comment.**

### **IDL Definition**

```
TCPA_RESULT TPM_CreateWrapKeyToPcr(
    [in, out] TCPA_AUTH* KeySlotAuth,
    [AUTH, in] UINT32 MaxWrapSize,
    [AUTH, in] UINT32 targetPCRSize,
    [AUTH, in, size_is(targetPCRSize)] BYTE* targetPCR,
    [AUTH, in] TCPA_ENCAUTH DataUsageAuth,
    [AUTH, in] TCPA_ENCAUTH DataMigrationAuth,
    [AUTH, in] TCPA_COMPOSITE_HASH targetPCRHash,
    [AUTH, in] TCPA_KEY_SLOT KeySlot,
    [AUTH, in] UINT16 Type,
    [AUTH, in] TCPA_KEYUSAGE KeyUsage,
    [AUTH, in] TCPA_DATAUSAGE DataUsage,
    [AUTH, in, out] TCPA_KEY keyInfo,
```

### **Type**

TCPA protected capability; the user must provide authorization to use the key indicated by keySlot.

### **Parameters**

Type	Name	Description
TCPA_AUTH	KeySlotAuth	Authorization to use key in keySlot. Session type MUST be OSAP.
UINT32	MaxWrapSize	The maximum size of the wrap area
UINT32	targetPCRSize	This SHALL be the size of the targetPCR parameter
BYTE*	targetPCR	This SHALL be a TCPA_PCR_SELECTION structure containing the list of the indexes of the PCRs that are to be reported.
TCPA_ENCAUTH	DataUsageAuth	Encrypted authorization data to be used with the new key.
TCPA_ENCAUTH	DataMigrationAuth	Encrypted data used to authorize migration of the newly generated key.
TCPA_COMPOSITE_HASH	targetPCRHash	This SHALL be the composite digest of the PCR indexes and values to which parameter blob is to be sealed. This must have been constructed according to the algorithm described in 10.4.5 using the target PCR values.
TCPA_KEY_SLOT	KeySlot	Public key used to wrap the generated key
UINT16	Type	Used for nonstandard keys.
TCPA_KEYUSAGE	KeyUsage	Indicates the usage of the key

TCPA_DATAUSAGE	DataUsage	Indicates how the data will be used
TCPA_KEY	keyInfo	The input structure contains all parameters except pubkey and privkey (which are NULL), to specify the size and type of the new key. The output structure also contains pubkey (the public part of the new key) and privkey (the wrapped private key).

### Descriptions

This command requires the encryption of two parameters. To create two XOR strings the caller takes the two nonces in use by the OSAP session and concatenates one nonce and the session shared secret and then hashes the result. The hash from the shared secret and the odd numbered hash (generated by the caller) encrypts the DataUsageAuth. The hash from the shared secret and the even numbered hash (generated by the TPM) encrypts the DataMigrationAuth.

### Actions

The TPM SHALL do the following:

- Validate the authorization to use the key pointed to by KeySlot. Return TPCA\_BADAUTH on any error.
- If the key in KeySlot is migratable and requested key is non-migratable
- Validate the key in KeySlot is a storage key
- Check the validity of the migration nonce in KeySlot
- Validate all other parameters.
- Any error on above checks return TPCA\_BAD\_PARAMETER
- Create the two XOR patterns by using the session key and the nonces for this transaction
- Decrypt the DataUsageAuth and DataMigrationAuth parameters
- Generate asymmetric key according to parameters sent
- Create the composite digest: **If the targetPCRSize parameter value is 0x00**, the TPM MUST return the error code TPCA\_BADINDEX. Otherwise, the TPM will compute a TPCA\_COMPOSITE\_HASH value, using the targetPCR TPCA\_PCR\_SELECTION structure, to fill the TPM\_SEALED\_DATA.digestAtCreation member variable. The TPM MUST compute this TPCA\_COMPOSITE\_HASH value using the targetPCR parameter as the input to the algorithm described in 10.4.5. The TPM MUST set the TPM\_SEALED\_DATA.IsSealedToPCR value to TRUE.

The TPM MUST check that the targetPCR parameter is a consistent TPCA\_PCR\_SELECTION structure. If not, the TPM MUST return the error code TPCA\_BADINDEX.

- Create a TPCA\_KEY structure using the key pointed to by KeySlot as the key for any encryptions
- Return the TPCA\_KEY structure in the KeyInfo parameter

Return Value	Description
TCPA_SUCCESS	Success
TCPA_FAIL	General error
TCPA_BAD_PARAMETER	One of the parameter was in error
TCPA_BADAUTH	The authorization to use keySlot failed

## 7.2.7 TSS\_WrapKey

### **Start of informative comment:**

The TSS\_WrapKey command creates a migratable blob for a key that has been presented externally. The creator of the key can prevent migration by the User by wrapping it with a non-migratable storage key and loading random data for the MigrationAuthorizationData. However, the internal bit will still be set as migratable. This allows delegation of a key without giving the delegator the right to further delegate. Because the key was created elsewhere, there is no need to return the PubKey of the key being wrapped, and because a public key is used for the wrapping, external to the TPM, there is no need for authorization data for the wrapping key to be passed.

### **End of informative comment.**

### **IDL Description**

```
TCPA_RESULT TSS_WrapKey(
    [in] UINT32 MaxWrapSize,
    [in] UINT32 KeyToWrapSize,
    [in] T CPA_SECRET DataMigrationAuth,
    [in] T CPA_SECRET DataUsageAuth,
    [in] T CPA_PUBKEY PubKey,
    [in] T CPA_KEYUSAGE KeyUsage,
    [in] T CPA_AUTHDATA_USAGE DataUsage,
    [in, size_is(KeyToWrapSize)] BYTE* KeyToWrap,
    [in, out] UINT32* WrapSize,
    [out, size_is(*WrapSize)] BYTE* Wrap);
```

### **Type**

TSS capability

### **Parameters**

Type	Name	Description
UINT32	MaxWrapSize	The maximum size of the Wrap area
UINT32	KeyToWrapSize	The size of the KeyToWrap parameter
TCPA_SECRET	DataMigrationAuth	The data migration secret
TCPA_SECRET	DataUsageAuth	The data usage secret
TCPA_PUBKEY	PubKey	Public key used to wrap the KeyToWrap parameter
TCPA_KEYUSAGE	KeyUsage	Tells if key is a storage or usage key
TCPA_AUTHDATA_USAGE	DataUsage	Sets the frequency authorization data is needed for this key.
BYTE*	KeyToWrap	External key being wrapped.
UINT32*	WrapSize	Size of wrapped data
BYTE*	Wrap	Result of wrapping key.

### **Descriptions**

A TSS function

### **Actions**

TSS\_WrapKey is used for wrapping up keys that were generated somewhere other than the TPM so that they can be used by the TPM. Such keys will always be migratable. Wrapping can be done entirely

outside the TPM, by software, hence this is a TSS function. Wrapping is always done by a public key, hence there is no need for authorization to perform this function.

Return Value	Description
TCPA_SUCCESS	Success.
TCPA_FAIL	Failure.

## 7.2.8 TSS\_WrapKeyToPcr

### ***Start of informative comment:***

The TSS\_WrapKeyToPcr command is similar to the TSS\_WrapKey command except that it has an additional requirement for authorization of use: a PCR value must match the value given at blob-creation time. Thus, TSS\_WrapKeyToPcr creates a migratable blob for a key that has been presented externally. Both authorization data and a given PCR value are set as part of the authorization requirement.

### ***End of informative comment.***

### **IDL Description**

```
TCPA_RESULT TSS_WrapKeyToPcr(
    [in] UINT32 MaxWrapSize,
    [in] UINT32 KeyToWrapSize,
    [in] UINT32 targetPCRSize,
    [in] TCPA_SECRET DataMigrationAuth,
    [in] TCPA_SECRET DataUsageAuth,
    [in] TCPA_COMPOSITE_HASH targetPCRHash,
    [in] TCPA_PUBKEY PubKey,
    [in] TCPA_KEYUSAGE KeyUsage,
    [in] TCPA_AUTHDATA_USAGE DataUsage,
    [in, size_is(targetPCRSize)] BYTE* targetPCR,
    [in, size_is(KeyToWrapSize)] BYTE* KeyToWrap,
    [in, out] UINT32* WrapSize,
    [out, size_is(*WrapSize)] BYTE* Wrap);
```

### **Type**

TSS capability

### **Parameters**

Type	Name	Description
UINT32	MaxWrapSize	The maximum size of the wrap parameter
UINT32	KeyToWrapSize	The size of the KeyToWrap parameter
UINT32	targetPCRSize	The size of the targetPCR parameter
TCPA_SECRET	DataMigrationAuth	The authorization value to permit use
TCPA_SECRET	DataUsageAuth	The authorization value to permit migration
TCPA_COMPOSITE_HASH	targetPCRHash	This SHALL be the composite digest of the PCR indexes and values to which parameter blob is to be sealed. This must have been constructed according to the algorithm described in 10.4.5 using the target PCR values.
TCPA_PUBKEY	PubKey	Public key used to wrap the passed key
TCPA_KEYUSAGE	keyUsage	Tells if key is a storage or signature key
TCPA_AUTHDATA_USAGE	DataUsage	Sets the frequency authorization data is needed for this key.
BYTE*	targetPCR	This SHALL be a TCPA_PCR_SELECTION structure containing the list of the indexes of the PCRs to which the blob parameter is to be sealed.



BYTE*	KeyToWrap	External key being wrapped. Normal format would be TCPA_KEY
UINT32*	WrapSize	Size of wrapped data
BYTE*	Wrap	Result of wrapping key.

**Actions**

TSS\_WrapKeyToPcr behaves much the same as TSS\_WrapKey.

It takes a key generated external to the TPM and wraps it with PubKey. Such keys are always migratable. Use of the key by the TPM is restricted to such time as the PCR value referred to is in the correct state and the correct authorization data is applied.

Return Value	Description
TCAP_SUCCESS	Success.
TCPA_FAIL	Failure.

## 7.2.9 TPM\_LoadKey

### **Start of informative comment:**

Before the TPM can use a key to either wrap, unwrap, bind, unbind, seal, unseal, sign or perform any other action, it needs to be present in the TPM. The TPM\_LoadKey function loads the key into the TPM for further use. It is assumed that upper level software provides management of key slots within the TPM. As such, the index of slot to load the key is passed as part of the authenticated parameter list.

The load command must maintain a record of whether any previous key in the key hierarchy was bound to a PCR.

### **End of informative comment.**

### IDL Description

```
TCPA_RESULT TPM_LoadKey(
    [in, out] TPCA_AUTH* ParentKeyAuth,
    [AUTH, in] TPCA_KEY_SLOT InKeySlot,
    [AUTH, in] TPCA_KEY_SLOT ParentKeySlot,
    [AUTH, in] UINT32 InKeyBlobSize,
    [AUTH, in] UINT32 pcrListSize,
    [AUTH, in, size_is(pcrListSize)] BYTE* pcrList,
    [AUTH, in] TPCA_PUBKEY pubkey,
    [AUTH, in] TPCA_PRIVKEY privKey);
```

### Type

TCPA protected capability; user must provide authorization to use the parent key pointed to by ParentKeySlot.

### Parameters

Type	Name	Description
TCPA_AUTH*	ParentKeyAuth	Authorization to use the storage key to decrypt the incoming key.
TCPA_KEY_SLOT	InKeySlot	Index to internal TPM slot where decrypted InkeyBlob is loaded.
TCPA_KEY_SLOT	ParentKeySlot	TPM slot index of parent key.
UINT32	InKeyBlockSize	Size of the InKeyBlob parameter
UINT32	pcrListSize	This SHALL be the size of the pcrList parameter
BYTE*	pcrList	This SHALL be the TPCA_PCR_SELECTION structure filled in with the PCR indices necessary to load the key
TCPA_PUBKEY	pubKey	This SHALL be the public portion of the key to be loaded
TCPA_PRIVKEY	privKey	This SHALL be the private portion of the key to be loaded

### Actions

The TPM SHALL perform the following steps:

- Validate the authorization to use the key in ParentKeySlot

- Extract the encrypted TCPA\_STORE\_ASYMKEY from privKey
- Decrypt TCPA\_STORE\_ASYMKEY using the key pointed to by ParentKeySlot
- Validate the integrity of pubKey and decrypted TCPA\_STORE\_ASYMKEY
- Load PCR indices. The TPM SHALL NOT check current PCR state during the TPM\_LoadKey command
- Perform any processing necessary to make TCPA\_STORE\_ASYMKEY key available for operations
- Load key and key information into slot pointed to by InKeySlot. Any previous occupant of InKeySlot is overwritten.
- Set InKeySlot.PCRParent to ParentKeySlot.PCRParent. If ParentKeySlot.IsWrappedToPCR is TRUE set InKeySlot.PCRParent to TRUE.

**If the decrypted InKeyBlob's IsWrappedToPCR parameter is TRUE,**

If pcrList.pcrCount is 0, the TPM MUST return the TCPA\_NO\_PCR\_INFO error.

Otherwise, the TPM MUST store the information contained in the pcrList parameter (the indices of the PCRs to which the key in InKeyBlob is wrapped) together with the key that results from the decryption of InKeyBlob. The PCR indices information will thereafter be available to any command that needs to check the PCR configuration before using the key.

Every time before the loaded key is used, the pcrList indices from TPM\_LoadKey and the PcrDigest from the key's TCPA\_STORE\_ASYMKEY structure MUST be used to verify that the current PCR state is correct. The TPM MUST ensure that the PCRs to which the key was sealed are the same as the PCRs' values that exist at the time of key usage. To do this, the TPM will compute a composite hash using the pcrList parameter as the input to the composite hashing algorithm (See 10.4.5).

If the resulting composite hash matches the PcrDigest from the key's TCPA\_STORE\_ASYMKEY structure, the TPM is permitted to use the key. Otherwise, if the composite hashes do not match, the TPM is NOT permitted to use the key in the current PCR state, and the TPM MUST return TCPA\_WRONGPCRVAL.

**If the decrypted InKeyBlob's IsWrappedToPCR parameter is FALSE,**

The TPM MUST ignore the pcrList parameter, and proceed with loading the key.

The TPM SHALL enforce the use of slot 0 for the SRK only. Attempts to load into slot 0 fail with TCPA\_FAIL.

Return Value	Description
TCPA_SUCCESS	Success.
TCPA_NOSPACE	No room to load key.
TCPA_FAIL	Failure.
TCPA_NO_PCR_INFO	The list of PCR indices to which the key is wrapped has not been provided
TCPA_WRONGPCRVAL	The named PCR value does not match the current PCR value.

### 7.2.10 TPM\_GetPubKey

**Start of informative comment:**

The owner of a key may wish to obtain the public key value from a loaded key. This information may have privacy concerns so the command must have authorization from the key owner.

**End of informative comment.**

#### IDL Description

```
TCPA_RESULT TPM_GetPubKey(
    [in, out] TCPA_AUTH* KeyAuth,
    [AUTH, in] TCPA_KEY_SLOT KeySlot,
    [AUTH, out] TCPA_PUBKEY* pubKey);
```

#### Type

TCPA protected capability; user must provide authorization to use the key pointed to by KeySlot.

#### Parameters

Type	Name	Description
TCPA_AUTH*	KeyAuth	Authorization to use the key in KeySlot
TCPA_KEY_SLOT	KeySlot	Index to internal TPM slot that contains public key
TCPA_PUBKEY*	pubKey	Public key of key loaded KeySlot

#### Actions

The TPM SHALL perform the following steps:

- Validate the authorization to use the key in KeySlot
- Create a TCPA\_PUBKEY structure and return

Return Value	Description
TCPA_SUCCESS	Success.
TCPA_FAIL	Failure.

## 7.2.11 TPM\_CreateMigrationBlob

### **Informative Comment:**

To migrate keys from one TPM to another for backup, upgrade or to clone a key on another platform, the TPM needs to create a data blob that another TPM can deal with. This is done by loading in a backup public key that will be used by the TPM to create a new data blob for a migratable key.

The TPM Owner does the selection and authorization of migration keys. The TPM Owner performs the selection and authorization at any time prior to the execution of TPM\_CreateMigrationBlob by performing the TPM\_AuthorizeMigrationKey command.

### **End of informative comment**

### **IDL Definition**

```
TCPA_RESULT TPM_CreateMigrationBlob(
    [in, out] TPCA_AUTH* KeyMigrateAuth,
    [AUTH, in] TPCA_KEY_SLOT keySlot,
    [AUTH, in] UINT32 BlobMaxSize,
    [AUTH, in] TPCA_MIGRATIONKEYAUTH migrationStructure,
    [AUTH, in] TPCA_PUBKEY MigrationWrapKey,
    [AUTH, in, out] UINT32* BlobSize,
    [AUTH, in, out] UINT32* RandomSize,
    [AUTH, out, size_is(*RandomSize)] BYTE* Random,
    [AUTH, out, size_is(*BlobSize)] BYTE* Blob);
```

### **Type**

TCPA protected capability; user MUST provide authorization to use the key pointed to by keySlot.

### **Parameters**

Type	Name	Description
TCPA_AUTH*	KeyMigrateAuth	Authorization to migrate the key in key slot. The authorization calculation MUST use the migration authorization data
TCPA_KEY_SLOT	keySlot	The slot containing the private key to be migrated
UINT32	BlobMaxSize	This SHALL be the maximum size of the output Blob parameter
TCPA_MIGRATIONKEY AUTH	migrationStructure	This SHALL be the migration key and authorization to use the migration key
TCPA_PUBKEY	MigrationWrapKey	This SHALL be the public key that is destination. This key MUST be a 2048 bit RSA key or higher.
UINT32*	BlobSize	This SHALL be the size of the output Blob parameter
UINT32*	RandomSize	This SHALL be the size of the Random parameter
BYTE*	Random	A random string used to hide the key being backed up from the backup authority. It is the responsibility of the caller to properly store and protect this value.
BYTE*	Blob	This SHALL be the encrypted and XOR TPCA_STORE_ASYMKEY structure

## Actions

The TPM SHALL perform the following actions:

- Validate that the authorization to migrate the key in keyslot. The validation MUST use the migrationAuthorization secret.
- Validate that the key in keySlot is not marked as non-migratable
- Calculate a digest of Migration.migrationKey and tpmProof and compare to Migration.digest
- Create m1 by filling in a TPCA\_MIGRATE\_ASYMKEY structure from key in keySlot.
- Create k1 and k2 by splitting the prime factor field from TPCA\_MIGRATE\_ASYMKEY.data into 2 parts. k1 is 20 bytes long, k2 contains the remainder of the prime factor.
- Create o1 (which SHALL be 229 bytes for a 2048 bit RSA key) by performing the OAEP encoding of m1 using OAEP parameters of
  - $m = m1$
  - $pHash$  = migration authorization (the field removed from TPCA\_STORE\_ASYMKEY to create TPCA\_MIGRATE\_ASYMKEY)
  - $seed = s1 = k1$
- Create r1 a random value from the TPM RNG. The size of r1 MUST be the size of o1. Return r1 in the Random parameter.
- Create x1 by XOR of o1 with r1
- Create o2 (which SHALL be 255 bytes) by performing the OAEP encoding of x1 using OAEP parameters of
  - $m = x1$
  - $p$  = "Migration Blob" a ASCII string null terminated
  - $seed = s2 = 20$  bytes from TPM RNG
- Create f1 by filling in a TPCA\_INTERNAL\_HDR structure.
- Create b1 by concatenating f1 and o2
- Encrypt b1 with the MigrationWrapKey

The TPM does not check the PCR values when migrating values locked to a PCR.

Return Value	Description
TCPA_SUCCESS	Success
TCPA_SHORTRANDOM	Random string not long enough
TCPA_KEYNOTLOADED	No Backup key loaded
TCPA_KEYNOTFOUND	Key to be backed up not found
TCPA_MIGRATEFAIL	Migration authorization failed
TCPA_FAIL	Other failure

## 7.2.12 TPM\_MigrateMigrationBlob

### **Informative Comment:**

This command allows changing of the key that is wrapping the encrypted migration blob. This operation must be a TPM command to allow a migration entity to prove that it is performing this operation inside a TPM.

### **End of informative comment**

### **IDL Definition**

```
TCPA_RESULT TPM_MigrateMigrationBlob(
    [in, out] T CPA_AUTH* keySlotAuth,
    [AUTH, in] T CPA_KEY_SLOT keySlot,
    [AUTH, in] UINT32 outBlobMaxSize,
    [AUTH, in] UINT32 inBlobSize,
    [AUTH, in] T CPA_PUBKEY ReWrapKey,
    [AUTH, in, out] UINT32* outBlobSize,
    [AUTH, in, size_is(inBlobSize)] BYTE* inBlob,
    [AUTH, out, size_is(*outBlobSize)] BYTE* outBlob);
```

### **Type**

TPM protected capability; user must provide authorization to use key pointed to by keySlot

### **Parameters**

Type	Name	Description
TCPA_AUTH*	keySlotAuth	Authorization to migrate the key in key slot. The authorization calculation MUST use the migration authorization data
TCPA_KEY_SLOT	keySlot	The slot containing the key that will unwrap the migration blob
UINT32	outBlobMaxSize	This SHALL be the maximum size of the outBlob parameter
UINT32	inBlobMax	This SHALL be the size of the inBlob parameter
TCPA_PUBKEY	ReWrapKey	This SHALL be the public key that will rewrap the inBlob parameter
UINT32*	outBlobSize	This SHALL be the size of the output Blob parameter
BYTE*	inBlob	This SHALL be the migration blob encrypted by the public key of the key pointed to by keySlot
BYTE*	outBlob	This SHALL be the migration blob encrypted by the ReWrapKey

### **Actions**

The TPM SHALL perform the following actions:

- Validate the authorization to use the key in keySlot
- Create d1 by decrypting the inBlob area using the key in keySlot
- Create o2 by removing T CPA\_INTERNAL\_HDR from d1
- Create m2 and p2hash by performing OAEP decoding of o2
- Verify that p2hash equals the SHA1 of "Migration Blob" an ASCII null terminated string
- Create outBlob by encrypting d1 with ReWrapKey

Return Value	Description
TCPA_SUCCESS	Success
TCPA_AUTHFAIL	Authorization to use the key pointed to by key slot was denied
TCPA_FAIL	Other failure



### 7.2.13 TPM\_LoadMigrationBlob

**Start of informative comment:**

This command takes a migration blob and creates a normal wrapped blob. The migrated blob must be loaded into the TPM using the normal TPM\_LoadKey function.

**End of informative comment.**

#### IDL Definition

```
TCPA_RESULT TPM_LoadMigrationBlob(
    [in, out] TPCA_AUTH* keySlotAuth,
    [AUTH, in] TPCA_KEY_SLOT keySlot,
    [AUTH, in] UINT32 outBlobMaxSize,
    [AUTH, in] UINT32 inBlobSize,
    [AUTH, in] UINT32 RandomSize,
    [AUTH, in, size_is(RandomSize)] BYTE* random,
    [AUTH, in, size_is(inBlobSize)] BYTE* inBlob,
    [AUTH, in, out] UINT32* outBlobSize,
    [AUTH, out, size_is(*outBlobSize)] BYTE* outBlob);
```

#### Type

TCPA protected capability; user must provide authorization

#### Parameters

Type	Name	Description
TCPA_AUTH*	keySlotAuth	Authorization to use the key in keySlot
TCPA_KEY_SLOT	keySlot	This SHALL be the pointer to the key to perform the unwrap
UINT32	outBlobMaxSize	This SHALL be the maximum size of the outBlob parameter
UINT32	inBlobSize	This SHALL be the size of the inBlob parameter
UINT32	randomSize	This SHALL be the size of the random parameter
BYTE*	random	This SHALL be the random string that provides the XOR for the area
BYTE*	inBlob	This SHALL be the encrypted and XOR migration blob
UINT32	outBlobSize	This SHALL be the size of the outBlob parameter
BYTE*	outBlob	This SHALL be the normally wrapped key blob

#### Action

The TPM SHALL perform the following:

- Validate the authorization to use the key in keySlot
- Create d1 by decrypting the inBlob area using the key in keySlot
- Create o2 by removing TPCA\_INTERNAL\_HDR from d1
- Create m2 and p2hash by performing OAEP decoding of o2
- Verify that p2hash equals the SHA1 of "Migration Blob" an ASCII null terminated string

- Create o1 by XOR m2 and random parameter
- Create m1, seed and pHash by OAEP decoding o1
- Create k1 by combining seed and the TCPA\_MIGRATE\_ASYMKEY.data field
- Create d2 a TCPA\_STORE\_ASYMKEY structure by inserting pHash as the migration authorization field. Set the TCPA\_STORE\_ASYMKEY.data field to k1
- Create outBlob by performing TCPA\_Internal\_Encrypt on d2 using the key in keySlot

Return Value	Description
TCPA_SUCCESS	Success.
TCPA_FAIL	Failure.

### 7.2.14 TPM\_AuthorizeMigrationKey

**Start of informative comment:**

To allow the TPM owner to specify which migration facility they will use and allow users to migrate information without further involvement with the TPM owner this command creates an authorization blob.

The TPM does no validation of the migration key. It is the responsibility of the TPM Owner to determine the validity of the key and if it is appropriate for use by the TPM.

**End of informative comment.**

#### IDL Definition

```
TCPA_RESULT TPM_AuthorizeMigrationKey(
    [in, out] TCPA_AUTH* ownerAuth,
    [AUTH, in] TCPA_PUBKEY migrationKey,
    [AUTH, in] UINT32 outBlobMaxSize,
    [AUTH, in, out] UINT32* outBlobSize,
    [AUTH, out, size_is(*outBlobSize)] BYTE* outBlob);
```

#### Type

TCPA protected capability; user must provide authorization from the TPM Owner

#### Parameters

Type	Name	Description
TCPA_AUTH*	ownerAuth	Authorization to use the TPM
TCPA_PUBKEY	migrationKey	This SHALL be the public key of the migration facility
UINT32	outBlobSize	This SHALL be the size of the outBlob parameter
BYTE*	outBlob	This SHALL be the normally wrapped key blob

#### Action

The TPM SHALL perform the following:

- Validate the authorization to use the TPM by the TPM Owner
- Create a digest value of migrationKey, TCPA\_PERSISTENT\_FLAGS.tpmProof
- Return the TCPA\_MIGRATIONKEYAUTH structure in the outblob parameter

Return Value	Description
TCPA_SUCCESS	Success.
TCPA_FAIL	Failure.

### 7.3 TPM Optional Functions: Maintenance

***Start of informative comment:***

Maintenance is different from backup/migration, because maintenance provides for the migration of both migratory and non-migratory data. Maintenance is an optional TPM function, but if a TPM enables maintenance, the maintenance capabilities in this specification are mandatory – no other migration capabilities shall be used. Maintenance necessarily involves the manufacturer of a Subsystem.

When maintaining computer systems, it is sometimes the case that a manufacturer or its representative needs to replace a Subsystem containing a TPM. Some manufacturers consider it a requirement that there be a means of doing this replacement without the loss of the non-migratable keys held by the original TPM.

The user needs assurance that the information is properly protected against interception or a hostile manufacturer therefore the creation of the maintenance information is fully defined. Since it is inherently NOT a process that can be performed between different models of systems, let alone different manufacturers the process which the manufacturer uses to install maintenance information is defined only at a high level.

Any maintenance process must have certain properties. Specifically, any migration to a replacement Subsystem must require collaboration between the Owner of the existing Subsystem and the manufacturer of the existing Subsystem. Further, the procedure must have adequate safeguards to prevent a non-migratable key being transferred to multiple Subsystems.

The maintenance capabilities TPM\_CreateMaintenanceArchive and TPM\_LoadMaintenanceArchive enable the transfer of all Protected Storage data from a Subsystem containing a first TPM (TPM<sub>1</sub>) to a Subsystem containing a second TPM (TPM<sub>2</sub>):

A manufacturer places a public key in non-volatile storage into its TPMs at manufacture time.

The Owner of TPM<sub>1</sub> uses TPM\_CreateMaintenanceArchive to create a maintenance archive that enables the migration of all data held in Protected Storage by TPM<sub>1</sub>. The Owner of TPM<sub>1</sub> must provide his or her authorization to the Subsystem. The TPM then creates the TPCA\_MAINTENANCE\_ASYMKEY structure and follows the process defined.

The XOR process prevents the manufacturer from ever obtaining plaintext TPM<sub>1</sub> data.

The additional random data provides a means to assure that a maintenance process cannot subvert archive data and hide such subversion.

The random mask can be generated by two methods, either using the TPM RNG or MGF1 on the TPM Owners authorization data.

The manufacturer takes the maintenance blob, decrypts it with its private key, and satisfies itself that the data bundle represents data from that Subsystem manufactured by that manufacturer. Then the manufacturer checks the endorsement certificate of TPM<sub>2</sub> and verifies that it represents a platform to which data from TPM<sub>1</sub> may be moved.

The manufacturer dispatches two messages.

The first message is made available to CAs, and is a revocation of the TPM<sub>1</sub> endorsement certificate.

The second message is sent to the Owner of TPM<sub>2</sub>, which will communicate the SRK, tpmProof and the manufacturers permission to install the maintenance blob only on TPM<sub>2</sub>

The Owner uses TPM\_LoadMaintenanceArchive to install the archive copy into TPM<sub>2</sub>, and overwrite the existing TPM<sub>2</sub>-SRK and TPM<sub>2</sub>-tpmProof in TPM<sub>2</sub>. TPM<sub>2</sub> overwrites TPM<sub>2</sub>-SRK with TPM<sub>1</sub>-SRK, and overwrites TPM<sub>2</sub>-tpmProof with TPM<sub>1</sub>-tpmProof.

Note that the command TPM\_KillMaintenanceFeature prevents the operation of TPM\_CreateMaintenanceArchive and TPM\_LoadMaintenanceArchive. This enables an Owner to block maintenance (and hence the migration of non-migratory data) either to or from a TPM.

It is required that a manufacturer takes steps that prevent further access of migrated data by TPM<sub>1</sub>. This may be achieved by deleting the existing Owner from TPM<sub>1</sub>, for example.

***End of informative comment.***

Any migration of non-migratory data protected by a Subsystem SHALL require the cooperation of both the Owner of that non-migratory data and the manufacturer of that Subsystem. That manufacturer SHALL NOT cooperate in a maintenance process unless the manufacturer is satisfied that non-migratory data will exist in exactly one Subsystem. A TPM SHALL NOT provide capabilities that support migration of non-migratory data unless those capabilities are described in the TCPA specification.

The maintenance feature MUST move the following

- TCPA\_KEY for SRK
- TCPA\_PERSISTENT\_FLAGS.tpmProof
- TPM Owners auth
- Hash of PUBEK

### 7.3.1 TPM\_CreateMaintenanceArchive

**Start of informative comment:**

This command creates the MaintenanceArchive. It can only be executed by the owner, and may be shut off with the TPM\_KillMaintenanceFeature command.

**End of informative comment.**

**IDL Definition**

```
TCPA_RESULT TPM_CreateMaintenanceArchive(
    [in, out] TCPA_AUTH* TpmOwnerAuth,
    [AUTH, in] UINT32 ArchiveMaxSize,
    [AUTH, in] UINT32 RandomMaxSize,
    [AUTH, in] BOOL GenerateRandom,
    [AUTH, in, out] UINT32* ArchiveSize,
    [AUTH, in, out] UINT32* randomSize,
    [AUTH, out, size_is(*randomSize)] BYTE* RandomData,
    [AUTH, out, size_is(*ArchiveSize)] BYTE* Archive);
```

**Type**

TCPA protected capability; user must provide authentication from the TPM Owner.

**Parameters**

Type	Name	Description
TCPA_AUTH	TpmOwnerAuth	Owner's authorization to make a maintenance backup.
UINT32	ArchiveMaxSize	Maximum size for archive.
UINT32	RandomMaxSize	Maximum size for the random parameter
BOOL	GenerateRandom	This SHALL indicate if TRUE that the TPM uses the RNG to create the random string. If FALSE the TPM uses the TPM Owner authorization to create the random string.
UINT32*	ArchiveSize	Size of archive being returned.
UINT32	randomSize	This SHALL be the size of the RandomData parameter, the MUST be a minimum of 256 bytes.
BYTE*	RandomData	Random data to XOR with result before encrypting with manufacturer's public key. Only returned when the GenerateRandom is TRUE.
BYTE*	Archive	Archive being returned.

**Actions**

Upon authorization being confirmed this command does the following:

- Validates that the TCPA\_PERSISTENT\_FLAGS.AllowMaintenance is TRUE.
- Validates the TPM Owner authorization.
- Create m1 by filling in a TCPA\_MAINTENANCE\_ASYMKEY structure using the SRK
- Create o1 (which SHALL be 208 bytes for a 2048 bit RSA key) by performing the OAEP encoding of m1 using OAEP parameters of
  - $m = m1$

- $P$  = TPM Owner authorization
  - $seed = s1 = 20$  bytes from the TPM RNG
- **If GenerateRandom = TRUE**
  - Create r1 by obtaining values from the TPM RNG. The size of r1 MUST be the same size as o1. Set RandomData parameter to r1
- **If GenerateRandom = FALSE**
  - Create r1 by applying MGF1 to the TPM Owner authorization data. The size of r1 MUST be the same size as o1. Set RandomData parameter to null.
- Create m2 by XOR of o1 and r1
- Create o2 (which SHALL be 255 bytes for a 2048 bit RSA key) by performing the OAEP encoding of m2 using OAEP parameters of
  - $m = m2$
  - $P = \text{PUBEK}$
  - $seed = s2 = 20$  bytes from the TPM RNG
- Create f1 by filling in a TPCA\_INTERNAL\_HDR structure.
- Create b1 by concatenating f1 and o2
- Encrypt b1 with the TPCA\_PERSISTENT\_FLAGS.ManufacturerPub

Return Value	Description
TCPA_SUCCESS	Success.
TCPA_FAIL	Failure.
TCPA_AUTHFAIL	TPM Owner authorization failed.
TCPA_DISABLED	The TPM is disabled
TCPA_DISABLED_CMD	The AllowMaintenance flag is FALSE

### 7.3.2 TPM\_LoadMaintenanceArchive

**Start of informative comment:**

This command loads in a Maintenance archive that has been massaged by the manufacturer to load into another TPM

**End of informative comment.**

**IDL Definition**

```
TCPA_RESULT TPM_LoadMaintenanceArchive(
    [in, out] TPCA_AUTH* TpmOwnerAuth,
    ... ) ;
```

**Type**

TCPA protected capability; user must provide authentication from the TPM Owner.

**Parameters**

Type	Name	Description
TCPA_AUTH*	TpmOwnerAuth	Authorization for the new TPM to replace its Storage Root Key with the one from the newPlatformDataBlob.
		Remaining parameters are manufacturer specific

**Actions**

The TPM SHALL perform the following when executing the command

- Validate the TPM Owner's authorization
- Validate that the maintenance information was sent by the TPME. The validation mechanism MUST use a strength of function that is at least the same strength of function as a digital signature performed using a 2048 bit RSA key.
- The packet MUST contain m2 as defined in 7.3.1
- Ensure that only the target TPM can interpret the maintenance packet. The protection mechanism MUST use a strength of function that is at least the same strength of function as a digital signature performed using a 2048 bit RSA key.
- Process the maintenance information and update the SRK and TPCA\_PERSISTENT\_FLAGS.tpmProof fields.

**Descriptions**

The maintenance mechanisms in the TPM MUST not require the TPM to hold a global secret. The definition of global secret is a secret value shared by more than one TPM.

The TPME is not allowed to pre-store or use unique identifiers in the TPM for the purpose of maintenance. The TPM MUST NOT use the endorsement key for identification or encryption in the maintenance process. The maintenance process MAY use a TPM Identity to deliver maintenance information to specific TPM's.

The maintenance process can only change the SRK and tpmProof fields.

The maintenance process can only access data in shielded locations where this data is necessary to validate the TPM Owner, validate the TPME and manipulate the blob

The TPM MUST be conformant to the TCPA specification, protection profiles and security targets after maintenance. The maintenance MAY NOT decrease the security values from the original security target.



The security target used to evaluate this TPM MUST include this command in the TOE.

Return Value	Description
TCPA_SUCCESS	Success.
TCPA_FAIL	Failure.
TCPA_AUTHFAIL	TPM Owner authorization failed.
TCPA_DISABLED	The TPM is disabled

### 7.3.3 TPM\_KillMaintenanceFeature

#### **Informative Comments:**

The KillMaintenanceFeature is a permanent action that prevents ANYONE from creating a maintenance archive. This action, once taken, is permanent until a new TPM Owner is set.

This action is to allow those customers who do not want the maintenance feature to not allow the use of the maintenance feature.

At the discretion of the Owner, it should be possible to kill the maintenance feature in such a way that the only way to recover maintainability of the platform would be to wipe out the root keys. This feature is mandatory in any TPM that implements the maintenance feature.

#### **End informative Comment**

#### **IDL Definition**

```
TCPA_RESULT TPM_KillMaintenanceFeature(
    [in, out] TCPA_AUTH* TpmOwnerAuth);
```

#### **Type**

TCPA protected capability; user must provide authentication from the TPM Owner.

#### **Parameters**

Type	Name	Description
TCPA_AUTH*	TpmOwnerAuth	This command takes only one parameter: authorization by the owner to shut off the maintenance feature.

#### **Actions**

- Validate the TPM Owner authorization
- Set the TCPA\_PERSISTANT\_FLAGS.AllowMaintenance flag to FALSE.

Return Value	Description
TCPA_SUCCESS	Success.
TCPA_FAIL	Failure.
TCPA_AUTHFAIL	TPM Owner authorization failed.
TCPA_DISABLED	The TPM is disabled

## 8. Cryptographic and Miscellaneous Functions

### 8.1 Introduction

This section describes the cryptographic functions and the miscellaneous functions that do not fit into any specific category.

### 8.2 Hash Operations

***Start of informative comment:***

The TSS must provide the support necessary to do a SHA-1 digest.

***End of informative comment.***

### 8.2.1 TSS\_HashAll

The TSS\_HashAll command is a TSS command that combines all three hash operations. The limitation of this command is that the area to hash must be contiguous.

#### IDL Definition

```
TCPA_RESULT TSS_HashAll(
    [in] UINT32 Algorithm,
    [in] UINT32 AlgParamSize,
    [in] UINT32 BufSize,
    [in, size_is(AlgParamSize)] BYTE* AlgParms,
    [in, size_is(BufSize)] BYTE* Buf,
    [out] TCPA_DIGEST* Digest);
```

#### Type

TSS function

#### Parameters

Type	Name	Description
UINT32	Algorithm	The algorithm to use
UINT32	AlgParamSize	Size of the algorithm buffer
UINT32	BufSize	The size of the buffer in bytes
BYTE*	Buffer	The buffer of information to the hash
BYTE*	AlgParms	The hash algorithm parameters
TCPA_DIGEST	Hash	The hash structure that keeps track of all state and operations

#### Actions

The TSS\_HashAll command calls TSS\_HashInit, TSS\_HashUpdate, and TSS\_HashFinal. This command hashes a contiguous buffer in only one call.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_FAIL	A critical internal error occurred.

## 8.2.2 TSS\_HashInit

The TSS\_HashInit command starts the hash process.

### IDL Definition

```
TCPA_RESULT TSS_HashInit(
    [in] UINT32 Algorithm,
    [in] UINT32 AlgParamSize,
    [in, size_is(AlgParamSize)] BYTE* AlgParms,
    [out] TSS_HASHHANDLE* HashHandle);
```

### Type

TSS function

### Parameters

Type	Name	Description
UINT32	Algorithm	The algorithm to use
UINT32	AlgParamSize	Size of the algorithm buffer
BYTE*	AlgParms	The hash algorithm parameters
TSS_HASHHANDLE	HashHandle	The handle that the TSS uses to locate the internal information regarding this hash operation

### Actions

The command validates the algorithm and parameters for the algorithm. There are no parameters for SHA1.

The command generates the structures and states to keep track of the hash operations.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_SIZE	There are too many open hash handles.
TCPA_FAIL	A critical internal error occurred.

### 8.2.3 TSS\_HashUpdate

The TSS\_HashUpdate command adds additional text to the hash.

#### IDL Definition

```
TCPA_RESULT TSS_HashUpdate(
    [in] TSS_HASHHANDLE HashHandle,
    [in] UINT32 BufSize,
    [in, size_is(BufSize)] BYTE* Buf);
```

#### Type

TSS function

#### Parameters

Type	Name	Description
TSS_HASHHANDLE	HashHandle	Handle to the hash structure
UINT32	BufSize	The size of the buffer in bytes.
BYTE*	Buf	The buffer of information to add to the hash.

#### Actions

The command locates the internal structures and state using the handle. The command adds the buffer of information to the hash. The TPM keeps the intermediate state of the hash as part of the internal structures.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_INVALID_HANDLE	Invalid hash handle.
TCPA_FAIL	A critical internal error occurred.

### 8.2.4 TSS\_HashFinal

The TSS\_HashFinal command completes the hash process.

#### IDL Definition

```
TCPA_RESULT TSS_HashFinal(  
    [in] TSS_HASHHANDLE HashHandle,  
    [out] TCPA_DIGEST* Digest);
```

#### Type

TSS function

#### Parameters

Type	Name	Description
UINT32	HashHandle	The size of the result in bytes.
TCPA_DIGEST	Hash	The result of the hash operation

#### Actions

The TSS\_HashFinal command takes the intermediate state and performs the final steps of the hash algorithm to obtain the output.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_INVALID_HANDLE	Invalid hash handle.
TCPA_FAIL	A critical internal error occurred.

### 8.3 HMAC Commands

***Start of informative comment:***

The TSS must provide the functionality to perform a HMAC calculation.

***End of informative comment.***

The TSS MUST support the HMAC using the SHA-1 hashing operation and protocol as defined by RFC 2104.

**Algorithms defined**

```
#define TSS_ALG_HMAC      0x00000002
```



### 8.3.1 TSS\_HMACAll

The TSS\_HMACAll command is a TSS command that combines all three HMAC operations. The limitation of this command is that the area to hash must be contiguous.

#### IDL Definition

```
TCPA_RESULT TSS_HMACAll(
    [in] UINT32 Algorithm,
    [in] UINT32 SecretSize,
    [in] UINT32 BufSize,
    [in] UINT32 AlgParmSize,
    [in, size_is(SecretSize)] BYTE* Secret,
    [in, size_is(BufSize)] BYTE* Buf,
    [in, size_is(AlgParmSize)] BYTE* AlgParms,
    [out] TCPA_DIGEST* HMAC);
```

#### Type

TSS function

#### Parameters

Type	Name	Description
UINT32	Algorithm	Algorithm to create the HMAC
UINT32	SecretSize	Size of the secret area
UINT32	BufSize	The size of the buffer to hash in bytes
UINT32	AlgParmSize	The size of the algorithm parameters in bytes
BYTE*	Secret	Secret value used in HMAC calculation
BYTE*	Buffer	The buffer of information to add to the hash.
BYTE*	AlgParms	The parameters for the HMAC operation
TCPA_DIGEST	HMAC	The resulting HMAC operation.

#### Actions

The TSS\_HMACAll command calls TSS\_HMACInit, TSS\_HMACUpdate, and TSS\_HMACFinal operations. This command is just for convenience.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_FAIL	A critical internal error occurred.

### 8.3.2 TSS\_HMACInit

The TSS\_HashInit command starts the HMAC process.

#### IDL Definition

```
TCPA_RESULT TSS_HMACInit(
    [in] UINT32 Algorithm,
    [in] UINT32 SecretSize,
    [in] UINT32 AlgParmSize,
    [in, size_is(SecretSize)] BYTE* Secret,
    [in, size_is(AlgParmSize)] BYTE* AlgParms,
    [out] TSS_HMACHANDLE* HmacHandle);
```

#### Type

TSS function

#### Parameters

Type	Name	Description
UINT32	Algorithm	Algorithm to create the HMAC
UINT32	SecretSize	Size of the secret area
UINT32	AlgParmSize	The size of the algorithm parameters in bytes
BYTE*	Secret	Secret value used in HMAC calculation
BYTE*	AlgParms	The parameters for the HMAC operation
TSS_HMACHANDLE*	HmacHandle	The handle for the HMAC internal structures and states

#### Actions

The TPM validates the algorithm and the algorithm parameters. The TPM then validates the authorization using the pubkey parameter and the authorization structure. The authorization type MUST be OSAP, as the authorization must be continued for the remaining HMAC operations.

The TPM creates the structures and states necessary to process the remaining HMAC operations and generates a handle to track the information.

The TPM has an internal limit as to the number of handles that may be open at one time, so the request for a new handle may fail if there is insufficient space available.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_FAIL	A critical internal error occurred.

### 8.3.3 TSS\_HMACUpdate

The TSS\_HashUpdate command adds additional information to the HMAC calculation.

#### Definition

```
TCPA_RESULT TSS_HMACUpdate(  
    [in] TSS_HMACHANDLE HmacHandle,  
    [in] UINT32 BufSize,  
    [in, size_is(BufSize)] BYTE* Buf);
```

#### Type

TSS function

#### Parameters

Type	Name	Description
TSS_HMACHANDLE	HmacHandle	The hash structure that keeps track of all state and operations.
UINT32	BufSize	The size of the buffer in bytes.
BYTE*	Buffer	The buffer of information to add to the hash.

#### Actions

The TSS locates the structures and states using the handle.

The TSS adds the information in the buffer to the hash and saves the intermediate hash state in the TSS.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_INVALID_HANDLE	Invalid HMAC handle.
TCPA_FAIL	A critical internal error occurred.

### 8.3.4 TSS\_HMACFinal

The TSS\_HashFinal command completes the HMAC process.

#### Definition

```
TCPA_RESULT TSS_HMACFinal(  
    [in] TSS_HMACHANDLE HmacHandle,  
    [out] TCPA_DIGEST* HMAC);
```

#### Type

TSS function

#### Parameters

Type	Name	Description
TSS_HMACHANDLE	HmacHandle	The hash structure that keeps track of all state and operations.
TCPA_DIGEST	HMAC	The resulting HMAC operation.

#### Actions

The TSS locates the structures and states using the handle.

The TSS then takes the intermediate state of the hash and performs the final steps of both the hash and HMAC process. The resulting HMAC value is returned in the HMAC parameter.

The TSS destroys all structures and states relating to the HMAC including the secret value.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_INVALID_HANDLE	Invalid HMAC handle.
TCPA_FAIL	A critical internal error occurred.

## 8.4 Key Certification

### 8.4.1 TPM\_CertifyKey

**Start of informative comment:**

The TPM\_CERTIFYKEY operation allows an identity key to certify the public portion of certain storage and signing keys. TPM\_CERTIFYKEY is allowed only for non-migratable keys. As such, it allows the TPM to make the statement “this key is held in a TCPA-shielded location, and it will never be revealed.” For this statement to have veracity, the Challenger must trust the policies used by the Privacy CA that issued the identity and the maintenance policy of the TPM manufacturer.

The key to be certified must be loaded before TPM\_CertifyKey is called.

TPM\_CERTIFYKEY and QUOTE are the only operations that use TPM identity keys, apart from those operations used to acquire identities.

**End of informative comment.**

#### IDL Definition

```
TCPA_RESULT TPM_CertifyKey(
    [in, out] TCPA_AUTH* IDauth,
    [in, out] TCPA_AUTH* CertifyKeyAuth,
    [AUTH, in] UINT32 BlobMaxSize,
    [AUTH, in] TCPA_KEY_SLOT KeyToCertify,
    [AUTH, in] TCPA_KEY_SLOT IdKey,
    [AUTH, in] TCPA_DIGEST ExternalData,
    [AUTH, in, out] UINT32* BlobSize,
    [AUTH, out, size_is(*BlobSize)] BYTE* Blob,
    [AUTH, out] TCPA_CERTIFY_INFO* SignHeader);
```

#### Type

TCPA protected capability; user must authorize the use of key pointed to by IdKey and the key pointed to by KeyToCertify.

#### Parameters

Type	Name	Description
TCPA_AUTH	IDAuth	Authorization data for the IdKey parameter
TCPA_AUTH	CertifyKeyAuth	Authorization data for the CertifyKeyAuth parameter
UINT32	BlobMaxSize	Maximum permissible size of the outgoing blob.
TCPA_KEY_SLOT	KeyToCertify	Key to be certified
TCPA_KEY_SLOT	IdKey	The key that will sign the new key. This MUST be a TPM identity key
TCPA_DIGEST	ExternalData	160-bits of externally supplied data (typically a nonce to prevent replay attacks).
UINT32*	BlobSize	Size of the outgoing blob
BYTE*	Blob	Pointer to memory that is to receive the signed data blob.
TCPA_CERTIFY_INFO	SignHeader	Information that defines how the signature was done

## Actions

The TPM validates that the key pointed to by idKey is an Identity Key.

The TPM verifies the authorization in IDAuth provides authorization to use the key pointed to by idKey.

The TPM verifies the authorization in CertifyKeyAuth provides authorization to use the key pointed to by KeyToCertify.

The TPM SHALL verify that the key pointed to by KeyToCertify can successfully perform an encryption and decryption of a nonce from the TPM RNG.

The TPM SHALL create a TPCA\_CERTIFY\_INFO (defined in section 4.19) structure from the key pointed to by KeyToCertify.

The TPM calculates the digest of the KeyToCertify public key and stores it in the pubkeyDigest field of the TPCA\_CERTIFY\_INFO structure.

The TPM assembles the externally provided data in the TPCA\_CERTIFY\_INFO structure's Data parameter.

**If the IsWrappedToPCR field of the key being certified is TRUE,**

The TPM MUST store the pcrDigest field of the key being certified in the DigestValue field of the TPM\_CERTIFY\_INFO structure.

**If the IsWrappedToPCR field of the key being certified is FALSE,**

The TPM MUST set the IsWrappedToPCR field of the TPM\_CERTIFY\_INFO structure to FALSE, and the TPM MUST set the pcrList.pcrCount field to 0, and the DigestValue field to 0.

The TPM then performs a TPM\_Internal\_Signature (See 8.16.2) on the signHeader parameter using the key pointed to by idKey. The resulting signed blob is returned in signatureBlob.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_INVALID_HANDLE	The key slot identifiers do not point to valid loaded keys
TCPA_BADPARAMETER	One or more parameters were bad.
TCPA_BUFSIZE	The output buffer is too small. *BlobSize is set to the size required.
TCPA_FAIL	An internal error occurred, or POST or a previous self-test failed.

## 8.5 Symmetric Encryption

***Start of informative comment:***

The symmetric-encryption entry point allows the TSS to perform symmetric encryption. This functionality allows a TPM implementation of symmetric encryption to have a standard entry point.

The TSS uses symmetric encryption in the certification of identities. Making a symmetric algorithm available in the TSS allows for creation of the certification messages entirely in the TSS.

Encryption is a three-step process: init, update and final. The init function performs the algorithm setup once and then allows the repeated use of the setup with the update function.

Decryption is a mirror of the encryption process.

The encryption algorithm works on complete blocks only. There is no padding done by this implementation.

***End of informative comment.***

### 8.5.1 TSS\_EncryptAll

The TSS\_EncryptAll command calls Init, Update and Final. The reason for this command is to provide a single call to the TSS for symmetric encryption.

#### IDL Definition

```
TCPA_RESULT TSS_EncryptAll(
    [in] UINT32 Algorithm,
    [in] UINT32 KeySize,
    [in] UINT32 RedSize,
    [in] UINT32 MaxBlackSize,
    [in] UINT32 AlgParmSize,
    [in, size_is(AlgParmSize)] BYTE* AlgParms,
    [in, size_is(KeySize)] BYTE* Key,
    [in, size_is(RedSize)] BYTE* RedArea,
    [in, out] UINT32* BlackSize,
    [out, size_is(*BlackSize)] BYTE* BlackArea);
```

#### Type

TSS function

#### Parameters

Type	Name	Description
UINT32	Algorithm	The encryption algorithm to use
UINT32	KeySize	Size of the key.
UINT32	RedSize	The size of the plaintext (red bits).
UINT32	MaxBlackSize	The maximum size of the output (black bit) area.
UINT32	AlgParmSize	Size of the algorithm parameters
BYTE*	AlgParms	Parameters for the algorithm
BYTE*	Key	The key for the encryption
BYTE*	RedArea	The plain text
UINT32*	BlackSize	The size of the output
BYTE*	BlackArea	The encrypted text

#### Actions

The command creates the TSS internal encryption handle and reserves any memory that the encryption process will require. The TSS fills in the TCPA\_ENCRYPT structure with the handle and the block size for the algorithm.

The setup process includes any processing of the key into the various structures that the encryption process will require.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_INVALID_HANDLE	Unable to create handle
TCPA_FAIL	A critical internal error occurred.



### 8.5.2 TSS\_EncryptInit

EncryptInit starts the encryption process.

#### IDL Definition

```
TCPA_RESULT TSS_EncryptInit(  
    [in] UINT32 Algorithm,  
    [in] UINT32 KeySize,  
    [in] UINT32 AlgParmSize,  
    [in, size_is(AlgParmSize)] BYTE* AlgParms,  
    [in, size_is(KeySize)] BYTE* Key,  
    [out] TCPA_ENCHANDLE* EncryptHandle);
```

#### Type

TSS function

#### Parameters

Type	Name	Description
UINT32	Algorithm	The encryption algorithm to use
UINT32	KeySize	Size of the key.
UINT32	AlgParmSize	Size of the algorithm parameters
BYTE*	AlgParms	Parameters for the algorithm
BYTE*	Key	The key for the encryption
TCPA_ENCHANDLE	EncryptHandle	The handle for the internal states and structures

#### Actions

The TSS validates the algorithm and any algorithm parameters. The TSS then creates the internal structures and states to manage the encryption process.

The TSS uses the key to perform any key setup tasks. The TSS may keep the key in internal memory or it may destroy the key.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_INVALID_HANDLE	Invalid hash handle.
TCPA_FAIL	A critical internal error occurred.

### 8.5.3 TSS\_EncryptUpdate

The EncryptUpdate command encrypts the block of redbits.

#### IDL Definition

```
TCPA_RESULT TSS_EncryptUpdate(
    [in] TCPA_ENCHANDLE EncryptHandle,
    [in] UINT32 RedSize,
    [in] UINT32 MaxBlackSize,
    [in, size_is(RedSize)] BYTE* RedArea,
    [in, out] UINT32* BlackSize,
    [out, size_is(*BlackSize)] BYTE* BlackArea);
```

#### Type

TSS function

#### Parameters

Type	Name	Description
TCPA_ENCHANDLE	EncryptHandle	The handle that points to the internal structures and state
UINT32	RedSize	The size of the input area (or red bits).
UINT32	MaxBlackSize	Maximum size of the output area
BYTE*	Redarea	The input area (red bits).
UINT32*	BlackSize	The size of the output area (or black bits).
BYTE*	Blackarea	The output area (black bits).

#### Actions

The TSS validates the handle and locates the structures and states for the encryption process. The input area must be the same size as the block for the encryption algorithm. The function encrypts the input and returns the encrypted area.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_INVALID_HANDLE	Invalid hash handle.
TCPA_FAIL	A critical internal error occurred.

### 8.5.4 TSS\_EncryptFinal

The EncryptFinal command completes the encryption process.

#### IDL Definition

```
TCPA_RESULT TSS_EncryptFinal(  
    [in] TCPA_ENCHANDLE EncryptHandle,  
    [in] UINT32 MaxBlackSize,  
    [in, out] UINT32* BlackSize,  
    [out, size_is(*BlackSize)] BYTE* BlackArea);
```

#### Type

TSS function.

#### Parameters

Type	Name	Description
TCPA_ENCHANDLE	EncryptHandle	The handle that points to the internal structures and state
UINT32	MaxBlackSize	Maximum size of the output area
UINT32*	BlackSize	The size of the output area (or black bits).
BYTE*	Blackarea	The output area (black bits).

#### Actions

The command completes the encryption process and deletes the encryption handle. All memory associated with the handle is deleted.

For most algorithms, there is no output for this command.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_INVALID_HANDLE	Invalid handle.
TCPA_FAIL	A critical internal error occurred.

### 8.5.5 TSS\_DecryptAll

The TSS\_DecryptAll command calls Init, Update and Final. The reason for this command is to provide a single call to the TPM if a provider is using the TPM for symmetric decryption.

#### IDL Definition

```
TCPA_RESULT TSS_DecryptAll(
    [in] UINT32 Algorithm,
    [in] UINT32 KeySize,
    [in] UINT32 BlackSize,
    [in] UINT32 MaxRedSize,
    [in] UINT32 AlgParmSize,
    [in, size_is(AlgParmSize)] BYTE* AlgParms,
    [in, size_is(KeySize)] BYTE* Key,
    [in, size_is(BlackSize)] BYTE* BlackArea,
    [in, out] UINT32* RedSize,
    [out, size_is(*RedSize)] BYTE* RedArea);
```

#### Type

TSS function

#### Parameters

Type	Name	Description
UINT32	Algorithm	The decryption algorithm to use
UINT32	KeySize	Size of the key.
UINT32	BlackSize	The size of the encrypted text (black bits).
UINT32	MaxRedSize	The maximum size of the output (red bit) area.
UINT32	AlgParmSize	Size of the algorithm parameters
BYTE*	AlgParms	Parameters for the algorithm
BYTE*	Key	The key for the decryption
BYTE*	BlackArea	The plain text
UINT32*	RedSize	The size of the output
BYTE*	RedArea	The decrypted text

#### Actions

The command creates the TPM internal decryption handle and reserves any memory that the decryption process will require. The TPM fills in the T CPA\_ENCRYPT structure with the handle and the block size for the algorithm.

The setup process includes any processing of the key into the various structures that the decryption process will require.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_INVALID_HANDLE	Unable to create handle
TCPA_FAIL	A critical internal error occurred.

### 8.5.6 TSS\_DecryptInit

DecryptInit starts the decryption process.

#### IDL Definition

```
TCPA_RESULT TSS_DecryptInit(  
    [in] UINT32 Algorithm,  
    [in] UINT32 KeySize,  
    [in] UINT32 AlgParmSize,  
    [in, size_is(AlgParmSize)] BYTE* AlgParms,  
    [in, size_is(KeySize)] BYTE* Key,  
    [out] TCPA_ENCHANDLE* DecryptHandle);
```

#### Type

TSS function

#### Parameters

Type	Name	Description
UINT32	Algorithm	The decryption algorithm to use
UINT32	KeySize	Size of the key.
UINT32	AlgParmSize	Size of the algorithm parameters
BYTE*	AlgParms	Parameters for the algorithm
BYTE*	Key	The key for the Decryption
TCPA_ENCHANDLE	DecryptHandle	The handle for the internal states and structures

#### Actions

The TSS validates the algorithm and any algorithm parameters. The TSS then creates the internal structures and states to manage the decryption process.

The TSS uses the key to perform any key setup tasks. The TSS may keep the key in internal memory or it may destroy the key.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_INVALID_HANDLE	Invalid hash handle.
TCPA_FAIL	A critical internal error occurred.

### 8.5.7 TSS\_DecryptUpdate

The DecryptUpdate command decrypts the block of black bits.

#### IDL Definition

```
TCPA_RESULT TSS_DecryptUpdate(
    [in] UINT32 DecryptHandle,
    [in] UINT32 BlackSize,
    [in, size_is(BlackSize)] BYTE* BlackArea,
    [in] UINT32 MaxRedSize,
    [in, out] UINT32* RedSize,
    [out, size_is(*RedSize)] BYTE* RedArea);
```

#### Type

TSS function.

#### Parameters

Type	Name	Description
TCPA_ENCHANDLE	DecryptHandle	The handle that points to the internal structures and state
UINT32	BlackSize	The size of the input area (or black bits).
UINT32	MaxRedSize	Maximum size of the output area
BYTE*	BlackArea	The input area (black bits).
UINT32*	RedSize	The size of the output area (or red bits).
BYTE*	RedArea	The output area (red bits).

#### Actions

The TSS validates the handle and locates the structures and states for the decryption process. The input area must be the same size as the block for the decryption algorithm. The function decrypts the input and returns the decrypted area.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_INVALID_HANDLE	Invalid hash handle.
TCPA_FAIL	A critical internal error occurred.

### 8.5.8 TSS\_DecryptFinal

The DecryptFinal command completes the Decryption process.

#### IDL Definition

```
TCPA_RESULT TSS_DecryptFinal(
    [in] TCPA_ENCHANDLE DecryptHandle,
    [in] UINT32 MaxRedSize,
    [in, out] UINT32* RedSize,
    [out, size_is(*RedSize)] BYTE* RedArea);
```

#### Type

TSS function.

#### Parameters

Type	Name	Description
TCPA_ENCHANDLE	DecryptHandle	The handle that points to the internal structures and state
UINT32	MaxRedSize	Maximum size of the output area
UINT32*	RedSize	The size of the output area (or red bits).
BYTE*	RedArea	The output area (red bits).

#### Actions

The command completes the decryption process and deletes the decryption handle. All memory associated with the handle is deleted.

For most algorithms, there is no output for this command.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_INVALID_HANDLE	Invalid handle.
TCPA_FAIL	A critical internal error occurred.

## 8.6 Digital Signatures

### 8.6.1 TPM\_Sign

**Start of informative comment:**

The Sign command signs a digest and returns the resulting digital signature. This command uses a properly authorized signature key.

**End of informative comment.**

**IDL Definition**

```
TCPA_RESULT TPM_Sign(
    [in, out] TCPA_AUTH* PubAuth,
    [AUTH, in] TCPA_KEY_SLOT KeySlot,
    [AUTH, in] TCPA_DIGEST Digest,
    [AUTH, in] UINT32 MaxSignSize,
    [AUTH, in, out] UINT32* SignSize,
    [AUTH, out] TCPA_VERSION* ver,
    [AUTH, out, size_is(*SignSize)] BYTE* SignArea);
```

**Type**

TCPA protected capability; user must provide authorization to use the keySlot parameter.

**Parameters**

Type	Name	Description
TCPA_AUTH	PubAuth	The authorization structure that authorizes the use of keySlot.
TCPA_KEY_SLOT	KeySlot	The keySlot identifier of a loaded key that can perform digital signatures.
TCPA_DIGEST	Digest	The digest value to sign
UINT32	MaxSignSize	The maximum size of the output buffer
UINT32*	SignSize	The length of the signArea
TCPA_VERSION	ver	This SHALL be a properly filled out version structure. See 4.5
BYTE*	SignArea	The resulting digital signature.

**Actions**

The TPM validates the authorization to use the key pointed to by keySlot. The TPM validates that the key pointed to by keySlot is allowed to perform digital signatures.

The TPM uses the Digest parameter as input to the PKCS#1 v2.0 RSAES\_OAEP encoding scheme.

The TPM encrypts the encoded area using the private key pointed to by keySlot.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_INVALID_HANDLE	Invalid handle.
TCPA_FAIL	A critical internal error occurred.



## 8.6.2 TSS\_VerifySignature

VerifySignature takes a hash and verifies the digital signature of the hash. VerifySignature only returns a TRUE or FALSE answer. The caller does not receive any information as to the reason for a failure.

### IDL Definition

```
TCPA_RESULT TSS_VerifySignature(  
    [in] UINT32 SigLen,  
    [in] TCPA_DIGEST Digest,  
    [in] TCPA_PUBKEY Pubkey,  
    [in, size_is(SigLen)] BYTE* Signature);
```

### Type

TSS capability

### Parameters

Type	Name	Description
UINT32	SigLen	The length of the signature area.
TCPA_DIGEST	Digest	Hash to verify
TCPA_PUBKEY	Pubkey	Identifier of key loaded in TPM
BYTE*	Signature	Signature blob to verify

### Actions

The TPM loads the signature blob. The TPM decrypts the signature blob. The TPM then removes the PKCS #1 padding and compares the digest parameter to the signed value. If they are the same the TPM returns TCPA\_SUCCESS otherwise the TPM returns TCPA\_FAIL. The TPM MUST NOT give out any additional information regarding the verification failure.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_FAIL	A critical internal error occurred.

## 8.7 Random Numbers

***Start of informative comment:***

The TPM has the ability to generate random numbers. This section merely exposes these numbers to allow entities outside of the TPM to use a random number.

The size of the output random area is only limited by the size of the parameter.

Some random number generator implementations are strengthened by adding entropy to the RNG at various intervals. The stir command allows those implementations to receive the entropy when it is available.

***End of informative comment.***

### 8.7.1 TPM\_GetRandom

GetRandom returns the next  $n$  bytes from the random number generator to the caller.

#### IDL Definition

```
TCPA_RESULT TPM_GetRandom(  
    [in] UINT32 BytesRequested,  
    [out, size_is(BytesRequested)] BYTE* Blob);
```

#### Type

TCPA protected capability.

#### Parameters

Type	Name	Description
UINT32	BytesRequested	The number of bytes to return. The maximum size is 256 bytes.
BYTE*	Blob	The output of the random bytes

#### Actions

This command fills in the random buffer with the next  $n$  bytes from the random number generator.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_FAIL	A critical internal error occurred.

### 8.7.2 TPM\_StirRandom

StirRandom adds entropy to the RNG state.

#### IDL Definition

```
TCPA_RESULT TPM_StirRandom(  
    [in] UINT32 BlobSize,  
    [in, size_is(BlobSize)] BYTE* Blob);
```

#### Type

TCPA protected capability.

#### Parameters

Type	Name	Description
UINT32	BlobSize	The size of the area
BYTE*	Blob	The area of data that will add entropy to the RNG state.

#### Actions

The TPM updates the state of the current RNG using the appropriate mixing function.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_FAIL	A critical internal error occurred.

## 8.8 Self Test

***Start of informative comment:***

The TPM has the ability to perform a self-test upon request. The test can be either a full test of the complete TPM, the same tests as done at startup or a test of an individual function.

The tests only return an T CPA\_SUCCESS or T CPA\_FAIL answer and never reveal why a certain test failed. Upon the failure of a self-test the TPM goes into failure mode and does not allow any additional operations to continue. There is a subset of operations that are permissible in the failure mode see the conformance document for details.

The TPM\_CertifySelfTest operation allows a requestor to validate that the self-test command executed and to trust the answer received.

The TPM\_CertifySelfTest requires the authorization of a key that can perform a digital signature.

The TPM\_CertifySelfTest is always the full self-test.

When the command fails for any reason, the command will not return a signature. The lack of a signature field returning to a Challenger is indication that some part of the process failed. The failure could be attacks against the signature or a failure in the TPM.

***End of informative comment.***

### 8.8.1 TPM\_SelfTestFull

**Start of informative comment:**

SelfTestFull tests all of the TCPA protected capabilities.

**End of informative comment.****IDL Definition**

```
TCPA_RESULT TPM_SelfTestFull(  
    [out] UINT32* manufacturerBigSecret);
```

**Type**

TCPA protected capability

**Parameters**

Type	Name	Description
UINT32*	manufacturerBigSecret	Manufacturer specific information

**Actions**

Performs the self-test for each functionality of the TPM.

Failure of only one function results in a failure for all and the TPM goes into failure mode.

Return Value	Description
TCPA_SUCCESS	The device passed all tests.
TCPA_FAIL	The device failed one or more tests.

## 8.8.2 TPM\_SelfTestStartup

**Start of informative comment:**

SelfTestFull performs the same tests that are done at startup.

**End of informative comment.**

**Definition**

```
TCPA_RESULT TPM_SelfTestStartup(
    [out] UINT32* manufacturerBigSecret);
```

**Type**

TCPA protected capability

**Parameters**

Type	Name	Description
UINT32*	manufacturerBigSecret	Manufacturer specific information

**Actions**

The TPM SHALL perform all required self-tests from section 10.8.1.

Failure of only one function results in a failure for all and the TPM goes into failure mode.

Return Value	Description
TCPA_SUCCESS	The device passed all tests.
TCPA_FAIL	The device failed one or more tests.

### 8.8.3 TPM\_CertifySelfTest

#### IDL Definition

```
TCPA_RESULT TPM_CertifySelfTest(
    [in, out] TCPA_AUTH* PubAuth,
    [AUTH, in] TCPA_KEY_SLOT keySlot,
    [AUTH, in] UINT32 MaxBlobSize,
    [AUTH, in, out] UINT32* BlobSize,
    [AUTH, out] UINT32* manufacturerBigSecret,
    [AUTH, out, size_is(*BlobSize)] BYTE* Blob);
```

#### Type

TCPA protected capability; user must provide authorization to use the keySlot parameter.

#### Parameters

Type	Name	Description
TCPA_AUTH	AuthData	Authorization to use the key pointed to by pubKey
TCPA_KEY_SLOT	keySlot	Slot where key that will perform signature is loaded
UINT32	MaxBlobSize	Maximum size of the blob.
UINT32*	BlobSize	Set to the size of the returned blob.
BYTE*	Blob	Pointer to memory that is to receive the signed data blob.
UINT32*	manufacturerBigSecret	Manufacturer specific information

#### Actions

The TPM SHALL perform TPM\_SelfTestFull.

After successful completion of the self-test the TPM then validates the authorization to use the key pointed to by keySlot.

The TPM creates a hash of the two nonce values associated with the authorization and signs the hash using the key identified by keySlot.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_BAD_PARAMETER	One or more parameters were bad.
TCPA_BUFSIZE	The output buffer is too small. *SigBlobActualSize is set to the size required.
TCPA_FAIL	An internal error occurred, or POST or a previous self-test failed.



## 8.9 Reset and Clear Operations

### ***Start of informative comment:***

Reset is the process of clearing all handles and sessions. The reset does not affect PCR values or volatile flag values that are set on TPM initialization. The reset does not affect the SRK or ownership values.

Clear is the process of returning the TPM to factory defaults. The clear commands need protection from unauthorized use and must allow for the possibility of changing Owners. The clear process has authorized commands and mechanisms to not allow the clear operation to occur.

The clear functionality performs the following tasks:

- Delete SRK. The deletion of the SRK includes the destruction of all protected storage areas below the SRK in the hierarchy. The areas below are not destroyed they just have no mechanism to be loaded anymore.
- All TPM volatile and non-volatile data is set to default value except the endorsement key pair. The clear includes the Owner-authorization data, so after performing the clear, the TPM has no Owner. The PCR values are undefined after a clear operation.
- The TPM shall return TPCA\_NOSRK until an Owner is set. After the execution of the clear command, the TPM must go through a power cycle to properly set the PCR values.

The Owner has ultimate control of when a clear occurs.

The Owner can perform the TPM\_OwnerClear command using the TPM Owner authorization. If the Owner wishes to disable this clear command and require physical access to perform the clear, the Owner can issue the TPM\_DisableOwnerClear command.

During the TPM startup processing anyone with physical access to the machine can issue the TPM\_ForceClear command. This command performs the clear. The TPM\_DisableForceClear disables the TPM\_ForceClear command for the duration of the power cycle. TSS startup code that does not issue the TPM\_DisableForceClear leaves the TPM vulnerable to a denial of service attack. The assumption is that the TSS startup code will issue the TPM\_DisableForceClear on each power cycle after the TSS determines that it will not be necessary to issue the TPM\_ForceClear command. The purpose of the TPM\_ForceClear command is to recover from the state where the Owner has lost or forgotten the TPM Ownership token.

The TPM\_ForceClear must only be possible when the issuer has physical access to the platform. The manufacturer of a platform determines the exact definition of physical access.

### ***End of informative comment.***

The TPM MUST support the reset operation. The reset operation clears all handles, sessions and volatile state machines. The reset MUST NOT affect the SRK, PCR and flags such as the flag set by TPM\_DisableForceClear.

The TPM MUST support the clear operations. The clear operation MUST perform the following actions:

- Perform a reset operation
- Delete the SRK
- Reset all non-volatile values to factory default except the endorsement key pair
- Return TPCA\_NOSRK until there is a proper execution of the ownership function

The TPM MUST support disabling the clear operations. After execution of the TPM\_DisableOwnerClear the TPM MUST require physical access to execute the TPM\_ForceClear. The TPM MUST support the TPM\_DisableForceClear to disable the TPM\_ForceClear command. The TPM\_DisableForceClear command MUST execute on each startup cycle to be effective.

### 8.9.1 TPM\_Reset

**Start of informative comment:**

In the case that a TSS driver loses track of the internal state of the TPM this command allows the driver to reset the TPM to a well-known state.

**End of informative comment.****IDL Definition**

```
TCPA_RESULT TPM_Reset();
```

**Type**

TCPA protected capability.

**Parameters**

None

**Actions**

The TPM deletes handles to all items in the TPM. This includes hash, HMAC and authorization sessions.

The TPM destroys all memory associated with any session. This includes secrets, nonces and state.

The TPM does not reset any PCR or DIR values.

The TPM does not reset any flags in the TCPA\_VOLATILE\_FLAGS structure.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_FAIL	An internal error occurred, or POST or a previous self-test failed

## 8.9.2 TPM\_Init

***Start of informative comment:***

TPM\_Init destroys most information on a TPM but will not unload keys that are both non-volatile and independent of keys bound to a PCR. This permits keys to be loaded for use during the next boot sequence.

***End of informative comment.*****Definition**

TPM\_Init

**Type**

TCPA protected capability that requires physical indication from the platform

**Parameters**

None

**Actions**

1. If a key in a key slot has the PCRParent indicator set to TRUE or the IsWrappedToPCR is TRUE or IsVolatile set to TRUE, the key **MUST** be unloaded. All other keys **MUST** remain in their key slots.
2. The TPM performs a TPM\_Reset.
3. The TPM performs all normal startup operations. These operations include resetting PCR values and all T CPA\_VOLATILE\_FLAGS.

The platform **MUST** be designed such that if the TPM\_Init signal is asserted the entire Platform **MUST** be initialized. This prevents, at least with a minimum effort, someone touching the TPM\_Init pin on the TPM and resetting only the TPM.

The TPM\_Init signal **MUST** have signaling qualifications appropriate for the required conformance and Protection Profile for the Platform.

### 8.9.3 TPM\_SaveState

**Start of informative comment:**

This warns a TPM to save some state information.

If a TPM's shielded storage is non-volatile, this command need have no effect.

If a TPM's shielded storage is volatile and the TPM alone is unable to detect the loss of external power in time to move data to non-volatile memory, this command should be presented before the TPM enters a low or no power state.

**End of informative comment.****Definition**

```
TCPA_RESULT TPM_SaveState();
```

**Type**

TCPA protected capability

**Parameters**

None

**Actions**

The contents of all PCRs MUST be preserved.

The contents of any key slot that is currently loaded SHOULD be preserved if the key's PCRParent indicator is FALSE and its IsWrappedToPCR indicator is FALSE and its IsVolatile indicator is FALSE. The contents of any key slot that is currently loaded MAY be preserved if its PCRParent indicator is TRUE or its IsWrappedToPCR indicator is TRUE or its IsVolatile indicator is TRUE.

Values MUST be preserved in their original shielded locations or as copies in other shielded locations.

Preserved values MUST be non-volatile.

If the parameter mirrored by a preserved value is altered by a protected capability other than TPM\_INIT, the preserved value MUST be declared invalid. If the parameter mirrored by any preserved value is altered by a protected capability other than TPM\_INIT, all preserved values MAY be declared invalid.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_BUSY	The TPM is too busy to do the command.

## 8.9.4 TPM\_Startup

### **Start of informative comment:**

Upon receipt of a TPM\_Init, the Platform may be in a power-on state or may be resuming from a suspended state. Some trusted entity will determine the startup state and must inform the TPM of the state.

### **End of informative comment.**

### **Definition**

```
TCPA_RESULT TPM_Startup(
    [in] T CPA_STARTUP_TYPE stType);
```

### **Type**

TCPA protected capability

### **Parameters**

Type	Name	Description
TCPA_STARTUP_TYPE	stType	This SHALL indicate the type of startup that is occurring

### **Actions**

TPM\_Startup MUST be generated by a trusted entity (the RTM or the TPM, for example).

TPM\_Startup MUST be presented to a TPM after a TPM\_Init command and prior to presentation of any other TPM command except TPM\_GetCapability.

If a TPM command, other than one or more TPM\_GetCapability commands, is executed after the TPM\_Init command and prior to the first TPM\_Startup command, the TPM MUST invalidate all preserved states, and enter an error state where only the TPM\_GetCapability command functions until another TPM\_Init command is issued.

### **If stType = T CPA\_ST\_CLEAR**

1. The TPM SHALL invalidate any preserved states and resume normal operation.

### **If stType = T CPA\_ST\_STATE**

1. The TPM SHALL take all necessary actions to ensure that all PCRs contain valid preserved values. If the TPM is unable to successfully complete these actions, it SHALL enter the TPM failure mode.
2. The TPM SHALL take all necessary actions to ensure that a key slot contains the preserved value of that key slot if the preserved value is valid and the preserved value's PCRParent indicator is FALSE and its IsWrappedToPCR indicator is FALSE and its IsVolatile indicator is FALSE. All other key slots MUST be unloaded. If the TPM is unable to successfully complete these actions, it SHALL enter the TPM failure mode.
3. The TPM SHALL invalidate any preserved values. If the TPM is unable to successfully complete this action, it SHALL enter the TPM failure mode
4. The TPM resumes normal operation. If the TPM is unable to resume normal operation, it SHALL enter the TPM failure mode.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_FAIL	The operation failed

### 8.9.5 TPM\_OwnerClear

***Start of informative comment:***

The OwnerClear command performs the clear operation under Owner authorization. This command is available until the Owner executes the DisableOwnerClear, at which time any further invocation of this command returns TCPA\_CLEAR\_DISABLED.

***End of informative comment.***

**IDL Definition**

```
TCPA_RESULT TPM_OwnerClear(
    [in, out] TCPA_AUTH* TpmOwnerAuth);
```

**Type**

TCPA protected capability; user must provide authorization as the TPM Owner.

**Parameters**

Type	Name	Description
TCPA_AUTH*	TpmOwnerAuth	Authorization data for the Owner

**Actions**

The TPM verifies that the TpmOwnerAuth properly authorizes the owner.

After owner verification the TPM then checks the status of the TCPA\_PERSISTENT\_FLAGS.DisableOwnerClear flag, if set the TPM returns TCPA\_CLEAR\_DISABLED.

The TPM executes the TPM\_Reset command. The TPM then destroys the SRK and any internal data associated with the SRK. The TPM then destroys the TPM Ownership data.

The result will be no Owner or SRK and the TPM is set to the state where it returns TCPA\_NOSRK.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_NOSRK	There is no Owner and no SRK value is available.
TCPA_CLEAR_DISABLED	The DisableOwnerClear command has turned off the ability for the OwnerClear command to execute.
TCPA_FAIL	An internal error occurred, or POST or a previous self-test failed.

## 8.9.6 TPM\_DisableOwnerClear

**Start of informative comment:**

The DisableOwnerClear command disables the ability to execute the TPM\_OwnerClear command permanently. Once invoked the only method of clearing the TPM will require physical access to the TPM.

**End of informative comment.****IDL Definition**

```
TCPA_RESULT TPM_DisableOwnerClear(  
    [in, out] TCPA_AUTH* TpmOwnerAuth);
```

**Type**

TCPA protected capability; user must provide authorization as the TPM Owner.

**Parameters**

Type	Name	Description
TCPA_AUTH	TpmOwnerAuth	Authorization data for the Owner

**Actions**

The TPM verifies that the TpmOwnerAuth properly authorizes the owner.

The TPM sets the TCPA\_PERSISTENT\_FLAGS.disableownerclear flag in the TPM that permanently disables the execution of the TPM\_OwnerClear command.

The only mechanism that can clear the TPM is the TPM\_ForceClear command. The TPM\_ForceClear command requires physical access to the TPM to execute.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_NOSRK	There is no Owner and no SRK value is available.
TCPA_FAIL	An internal error occurred, or POST or a previous self-test failed.

### 8.9.7 TPM\_ForceClear

***Start of informative comment:***

The ForceClear command performs the Clear operation under physical access. This command is available until the execution of the DisableForceClear, at which time any further invocation of this command returns TCPA\_CLEAR\_DISABLED.

***End of informative comment.***

**IDL Definition**

```
TCPA_RESULT TPM_ForceClear();
```

**Type**

TCPA protected capability; there must be some evidence of physical access to the platform present for the TPM to verify.

**Parameters**

None

**Actions**

The TPM checks for a prior execution of the TPM\_DisableForceClear command. If executed, the TPM will return TCPA\_CLEAR\_DISABLED.

After verification of physical access, the TPM performs a clear operation that has the same result as the TPM\_OwnerClear. The execution the result of this command is exactly like the TPM\_OwnerClear.

The implementation of this command is a manufacturer option. The evidence of physical access could be done by setting a pin high on a chip, or by sending special bus cycles or by any other mechanism that provides evidence of physical access.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_NOSRK	There is no Owner and no SRK value is available.
TCPA_CLEAR_DISABLED	The DisableOwnerClear command has turned off the ability for the OwnerClear command to execute.
TCPA_FAIL	An internal error occurred, or POST or a previous self-test failed.



### 8.9.8 TPM\_DisableForceClear

**Start of informative comment:**

The DisableForceClear command disables the execution of the ForceClear command until the next startup cycle. Once this command is executed, the TPM\_ForceClear is disabled until another startup cycle is run.

**End of informative comment.****IDL Definition**

```
TCPA_RESULT TPM_DisableForceClear();
```

**Type**

TCPA protected capability.

**Parameters**

None

**Actions**

The TPM sets the TCPA\_VOLATILE\_FLAGS.disableforceclear flag in the TPM that disables the execution of the TPM\_ForceClear command.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_FAIL	An internal error occurred, or POST or a previous self-test failed.

## 8.10 The GetCapability Commands

### ***Start of informative comment:***

The TPM has numerous capabilities that a remote entity may wish to know about. These items include support of algorithms, key sizes, protocols and vendor-specific additions. The GetCapability command allows the TPM to report back to the requestor what type of TPM it is dealing with.

There are two variations of the GetCapability command: one that provides a signed response and one that merely returns the answer without an accompanying signature. The information in each is the same except for the inclusion or absence of a digital signature.

The request for information requires the requestor to specify which piece of information that is required. The request does not allow the “merging” of multiple requests and returns only a single piece of information.

In failure mode the TPM can only return manufactures name, TPM model and TPM version.

### ***End of informative comment.***

The TPM MUST NOT return any information that identifies an individual TPM in the any GetCapability command.

### **IDL Definitions**

```
#define TPM_CAP_ORD                0x00000001
#define TPM_CAP_STAT               0x00000002
#define TPM_CAP_KEY                0x00000003
#define TPM_CAP_VENDOR             0x80000000

#define TPM_CAP_STAT_MAINT         0x00000301
#define TPM_CAP_CLEAR_OWNER        0x00000302
#define TPM_CAP_CLEAR_FORCE        0x00000303
```

### 8.10.1 TPM\_GetCapability

#### IDL Definition

```
TCPA_RESULT TPM_GetCapability(
    [in] UINT32 CapArea,
    [in] UINT32 SubCap,
    [in] UINT32 MaxRespSize,
    [in, out] UINT32* RespSize,
    [out, size_is(*RespSize)] BYTE* Resp);
```

#### Type

TCPA protected capability

#### Parameters

Type	Name	Description
UINT32	CapArea	Area to for request
UINT32	SubCap	Further definition of what information is being requested
UINT32	MaxRespSize	The maximum size of the response area
UINT32	RespSize	The size of the capability response.
BYTE*	Resp	The actual response

#### Actions

The TPM validates the capArea and subCap indicators. If the information is available, the TPM creates the response field and fills in the actual information.

capArea	subCap	Result
TPM_CAP_ORD	Command ordinal	Boolean value. TRUE TPM supports ordinal FALSE no support
TPM_CAP_STAT	TPM_ALG_XXX TPM_PRT_XXX TPM_ENC_XXX	Boolean value. TRUE TPM supports item, FALSE no support
TPM_CAP_STAT	TPM_CAP_STAT_MAINT	Boolean value. TRUE maintenance flag ON, FALSE maintenance flag OFF
TPM_CAP_STAT	TPM_CAP_CLEAR_OWNER	Boolean value. TRUE owner clear flag ON, FALSE owner clear flag OFF
TPM_CAP_STAT	TPM_CAP_CLEAR_FORCE	Boolean value. TRUE force clear flag ON, FALSE force clear flag
TPM_CAP_STAT	TPM_CAP_STAT_PCR	UINT32 value. Returns the count of PCR registers
TPM_CAP_STAT	TPM_CAP_STAT_DIR	UINT32 value. Returns the count of DIR registers.
TPM_CAP_STAT	TPM_CAP_STAT_VENDOR	UINT32 value. Identifier of the TPM manufacturer.
TPM_CAP_STAT	TPM_CAP_STAT_VER	TCPA_VERSION structure.
TPM_CAP_KEY	key slot number	TCPA_KEY_INFO. This structure contains information regarding the key.

		information regarding the key.
--	--	--------------------------------

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_BAD_PARAMETER	One or more parameters were bad.
TCPA_FAIL	An internal error occurred, or POST or a previous self-test failed.

## 8.10.2 TSS\_GetCapability

### IDL Definition

```
TCPA_RESULT TSS_GetCapability(
    [in] UINT32 CapArea,
    [in] UINT32 SubCap,
    [in] UINT32 MaxRespSize,
    [in, out] UINT32* RespSize,
    [out, size_is(*RespSize)] BYTE* Resp);
```

### Type

TSS function

### Parameters

Type	Name	Description
UINT32	CapArea	Partition of capabilities to be interrogated
UINT32	SubCap	Further definition of what information is being requested
UINT32	MaxRespSize	The maximum size of the response area
UINT32	RespSize	The size of the capability response.
BYTE*	Resp	The actual response

### Actions

The TPM validates the capArea and subCap indicators. If the information is available, the TPM creates the response field and fills in the actual information.

capArea	subCap	Result
TSS_STAT	TSS_RURDY	BOOL – TRUE TPM will accept commands, FALSE TPM is busy

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_BAD_PARAMETER	One or more parameters were bad.
TCPA_FAIL	An internal error occurred, or POST or a previous self-test failed.

### 8.10.3 TPM\_GetCapabilitySigned

**Start of informative comment:**

TPM\_GetCapabilitySigned takes the same input as TPM\_GetCapability the only difference is that the response area has a digital signature to validate the answer.

**End of informative comment.**

**IDL Definition**

```
TCPA_RESULT TPM_GetCapabilitySigned(
    [in, out] T CPA_AUTH* SigningKeySlotAuth,
    [AUTH, in] T CPA_KEY_SLOT SigningKeySlot,
    [AUTH, in] UINT32 CapArea,
    [AUTH, in] UINT32 SubCap,
    [AUTH, in] UINT32 MaxRespSize,
    [AUTH, in] UINT32 MaxSignSize,
    [AUTH, in, out] UINT32* RespSize,
    [AUTH, in, out] UINT32* SignSize,
    [AUTH, out] T CPA_VERSION* ver,
    [AUTH, out, size_is(*RespSize)] BYTE* Resp,
    [AUTH, out, size_is(*SignSize)] BYTE* Sign);
```

**Type**

TCPA protected capability; the user must supply authorization to use of parameter SigningKeySlot.

**Parameters**

Type	Name	Description
TCPA_AUTH	SigningKeySlotAuth	Authorization to use SigningKeySlot
TCPA_KEY_SLOT	SigningKeySlot	Slot containing signature key.
UINT32	CapArea	Partition of capabilities to be interrogated
UINT32	SubCap	Further definition of what information is being requested
UINT32	MaxRespSize	The maximum size of the response area
UINT32	MaxSignSize	The maximum size of the signature area
UINT32*	RespSize	The size of the capability response.
UINT32*	SignSize	The size of the signature area
TCPA_VERSION	ver	This SHALL be the current version, see 4.5
BYTE*	Resp	The response area as set by the capability response
BYTE*	Sign	The signature of the response

**Actions**

The TPM interprets the capArea and subCap fields to determine what the response should be. This is the same processing as done in TPM\_GetCapability.

With the response field available, the TPM validates the authorization to use the key pointed to by pubKey. After validation the TPM creates a digital signature of the response field and puts the resulting signature block in the sign parameter.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_KEYNOTFOUND	The PUBKEY of the key to be signed is not known to the TPM.
TCPA_BAD_PARAMETER	One or more parameters were bad.
TCPA_BUFSIZE	The output buffer is too small. *SigBlobActualSize is set to the size required.
TCPA_FAIL	An internal error occurred, or POST or a previous self-test failed.

## 8.11 Audit Commands

***Start of informative comment:***

The TPM and TSS need to be able to report a log of events. The log uses the same paradigm as the PCRs, the TPM keeps a PCR value that extends for each log event, and the TSS maintains the log entries for Challengers to review.

The Owner has the ability to set which functions generate an audit event and to change which functions generate the event at any time.

The status of the audit generation is not seen as sensitive information and so the command to determine the status of the generation is not an authorized command.

***End of informative comment.***

Each command ordinal in non-volatile TPM memory has an indicator if executing the command will result in the generation of an audit event.

The audit event includes the command ordinal and the return code from the command.



### 8.11.1 TPM\_GetAuditEvent

The TPM uses this command to get the audit information from the TPM.

#### IDL Definition

```
TCPA_RESULT TPM_GetAuditEvent(  
    [out] UINT32* CmdOrd,  
    [out] UINT32* Returncode,  
    [out] TCPA_DIGEST* Digest);
```

#### Type

TCPA protected capability.

#### Parameters

Type	Name	Description
UINT32*	CmdOrd	Ordinal of the last command executed.
UINT32*	Returncode	The return code for the last command executed
TCPA_DIGEST*	Digest	The running log of all audited events.

#### Actions

The TPM returns the ordinal of the last audited command. The TPM also returns the value of the running digest.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_FAIL	An internal error occurred, or POST or a previous self-test failed.

### 8.11.2 TSS\_GetAuditLog

Get the log from the TSS.

#### IDL Definition

```
TCPA_RESULT TSS_GetAuditLog(  
    [in] UINT32 MaxLogSize,  
    [in, out] UINT32* LogSize,  
    [out] TCPA_VERSION* Version,  
    [out] UINT32* EventCount,  
    [out, size_is(*LogSize)] BYTE* Log);
```

#### Type

TSS function.

#### Parameters

Type	Name	Description
UINT32	MaxLogSize	The maximum size of the output area
UINT32*	LogSize	The size of the log area
TCPA_VERSION*	Version	The version of the audit log.
UINT32*	EventCount	The count of events in the log
BYTE*	Log	The actual log entries. Each entry in the log is a TCPA_LOG_EVENT structure.

#### Actions

The TSS returns all log events in the log file.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_FAIL	An internal error occurred, or POST or a previous self-test failed.

### 8.11.3 TPM\_SetOrdinalAuditStatus

Set the audit flag for a given ordinal. This command requires the authorization of the TPM Owner.

#### IDL Definition

```
TCPA_RESULT TPM_SetOrdinalAuditStatus(  
    [in, out] TCPA_AUTH* TpmOwnerAuth,  
    [AUTH, in] UINT32 Ordinal,  
    [AUTH, in] BOOL* State);
```

#### Type

TCPA protected capability; the user must show authorization from the TPM Owner to execute the command.

Type	Name	Description
TCPA_AUTH	TpmOwnerAuth	TPM Owner authentication
UINT32	Ordinal	The ordinal to set the audit event handling
BOOL	State	The state of the ordinal's audit flag, where TRUE = auditing, FALSE = not auditing.

#### Actions

The TPM authenticates the command using the TPM Owner authentication. If authentication unsuccessful the TPM returns TCPA\_FAIL.

The TPM sets the state of the non-volatile flag for the given ordinal to the indicated state. The TPM also returns the state in the response.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_FAIL	An internal error occurred, or POST or a previous self-test failed.

### 8.11.4 TPM\_GetOrdinalAuditStatus

Get the status of the audit flag for the given ordinal.

#### IDL Definition

```
TCPA_RESULT TPM_GetOrdinalAuditStatus(  
    [in] UINT32 Ordinal,  
    [out] BOOL* State);
```

#### Type

TCPA protected capability.

#### Parameters

Type	Name	Description
UINT32	Ordinal	The ordinal to report the status on.
BOOL	State	The state of the ordinal's audit flag, where TRUE = auditing, FALSE = not auditing.

#### Actions

The TPM returns the Boolean value for the given ordinal.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_FAIL	An internal error occurred, or POST or a previous self-test failed.

## 8.12 Enabling Ownership

### ***Informative comment***

The purpose of these capabilities is to enable and disable the process of taking ownership of a TPM.

The process of enabling and disabling ownership uses a non-volatile flag `TCPA_PERSISTENT_FLAGS.ownership`. If the `TCPA_PERSISTENT_FLAGS.ownership` flag is `TRUE`, the TPM will not permit the “take ownership” command to operate. If the flag is `FALSE`, it has no effect on any other capability. See section 4.22.1 for the `TCPA_PERSISTENT_FLAGS.ownership` flag.

This enable-Ownership command on its own does not provide the necessary privacy controls for a TPM. It should be considered together with the operation of the enable/disable commands of section 8.13 and the activate/deactivate commands of section 8.14. The activate/deactivate commands are weaker forms of the enable/disable commands, in that they permit the process of taking Ownership of a TPM. The enable-Ownership, enable/disable, and activate/deactivate commands together permit the taking of TPM Ownership without the risk of inadvertent use of a TPM. See section 2.6.

Physical presence authorizes the changing of the `TCPA_PERSISTENT_FLAGS.ownership` flag.

A remote entity must not be able to change the setting of the `TCPA_PERSISTENT_FLAGS.ownership` flag.

***End of informative comment.***

### 8.12.1 TPM\_SetOwnerInstall

#### IDL Definition

```
TCPA_RESULT TPM_SetOwnerInstall (  
    [in] BOOL* State);
```

#### Type

TCPA protected capability; there must be some evidence of physical access present for the TPM to verify.

#### Parameters

Type	Name	Description
BOOL	State	State to set ownership flag to

#### Action

If the TPM has a current owner, this command immediately returns with TCPA\_SUCCESS.

The TPM validates the assertion of physical access. The TPM then sets the value of TCPA\_PERSISTENT\_FLAGS.ownership to the value in state.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_FAIL	A critical system error occurred

## 8.13 Enabling a TPM

### ***Informative comment***

The purpose of these capabilities is to enable and disable a TPM without destroying secrets protected by the TPM.

The process of enabling and disabling a TPM uses the non-volatile TCPA\_PERSISTENT\_FLAGS.disable flag. When set to TRUE, the TPM will reject most commands. Note, however, that a disabled TPM never disables the “extend” capability. This is necessary in order to ensure that the PCR values in a TPM are always up-to-date. If the flag is FALSE, it has no effect on other capabilities. See section 4.22.1 for the full effects of the TCPA\_PERSISTENT\_FLAGS.disable flag.

These enable/disable commands on their own do not provide the necessary privacy controls for a TPM. They should be considered together with the operation of the enable\_ownership command of section 8.12 and the activate/deactivate commands of section 8.14. The activate/deactivate commands are weaker forms of the enable/disable commands, in that they permit the process of taking Ownership of a TPM. The enable-Ownership, enable/disable, and activate/deactivate commands together permit the taking of TPM Ownership without the risk of inadvertent use of a TPM. See section 2.6.

There are two mechanisms to change the status of the TCPA\_PERSISTENT\_FLAGS.disable flag. The first mechanism is by using the owner-authenticated command TPM\_OwnerSetDisable. The second uses the two commands TPM\_PhysicalEnable and TPM\_PhysicalDisable. These two commands require the assertion of physical presence. TPM\_PhysicalEnable must be incapable of subversion by software.

### ***End of informative comment.***

### 8.13.1 TPM\_OwnerSetDisable

#### IDL Definition

```
TCPA_RESULT TPM_OwnerSetDisable(  
    [in, out] TCPA_AUTH* TpmOwnerAuth,  
    [AUTH, in, out] BOOL* State);
```

#### Type

TCPA protected capability; the user must provide authorization.

#### Parameters

Type	Name	Description
TCPA_AUTH	TpmOwnerAuth	Authorization from TPM Owner
BOOL	State	State to set disable flag to

#### Action

The TPM SHALL authenticate the command as coming from the TPM Owner. If unsuccessful, the TPM SHALL return TCPA\_BAD\_AUTH.

The TPM SHALL set the TCPA\_PERSISTENT\_FLAGS.disable flag to the value in the state parameter.

The TPM SHALL return the value of the flag in the response.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_BAD_PARAMETER	Parameter was not recognized
TCPA_FAIL	A critical system error occurred



### 8.13.2 TPM\_PhysicalDisable

#### IDL Definition

```
TCPA_RESULT TPM_PhysicalDisable();
```

#### Type

TCPA protected capability; there must be some evidence of physical access present for the TPM to verify.

#### Parameters

None

#### Action

The TPM SHALL set the TCPA\_PERSISTENT\_FLAGS.disable value to TRUE. The TPM while executing this command MUST obtain assurance from a physical method that operation of this command is authorized.

The TPM manufacturer MAY implement this command not as a response to a message block but as a response to a physical action, for instance, the acceptance of a special bus cycle or setting a pin high.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_FAIL	A critical system error occurred

### 8.13.3 TPM\_PhysicalEnable

**Definition**

TPM\_PhysicalEnable

**Type**

TCPA protected capability; there must be some evidence of physical access present for the TPM to verify.

**Parameters**

None

**Action**

The TPM SHALL set the TCPA\_PERSISTENT\_FLAGS.disable value to FALSE. The TPM while executing this command MUST obtain assurance from a physical method that operation of this command is authorized.

The TPM manufacturer MUST implement this command as a response to a physical action, for instance, the acceptance of a special bus cycle or setting a pin high.

The platform SHALL be incapable of subverting this command.

There is no IDL message block defined for this command.

## 8.14 Activating a TPM

### ***Informative comment***

The purpose of these capabilities is to activate and deactivate a TPM without destroying secrets protected by the TPM. This is subtly different from enabling and disabling a TPM.

An inactive TPM permits more commands to operate than does a disabled TPM. In particular, an inactive TPM does not block the enabling/disabling of a TPM and the process of taking ownership of the TPM. An inactive TPM never prevents the “extend” capability from operating. This is necessary in order to ensure that the PCR values in a TPM are always up-to-date.

These activate/deactivate commands on their own do not provide the necessary privacy controls for a TPM. They should be considered together with the operation of the enable\_Ownership commands of section 8.12 and the enable/disable commands of section 8.13. The enable/disable commands are stronger forms of the activate/deactivate commands, in that they do not permit the process of taking Ownership of a TPM. The enable-Ownership, enable/disable, and activate/deactivate commands together permit the taking of TPM Ownership without the risk of inadvertent use of a TPM. See section 2.6.

There are TWO deactivated flags, one volatile and one non-volatile. At switch-on, the volatile flag is set to the same state as the non-volatile flag. Altering the non-volatile flag requires physical presence at the platform. The volatile flag can be set without authentication, but its effect lasts only until the platform is rebooted.

See section 4.22.1 for the full effect of the TCPA\_PERSISTENT\_FLAGS.deactivated flag. See section 4.22.2 for the full effects of the TCPA\_VOLATILE\_FLAGS.deactivated flag.

***End of informative comment.***

### 8.14.1 TPM\_PhysicalSetDeactivated

#### IDL Definition

```
TCPA_RESULT TPM_PhysicalSetDeactivated(  
    [in] BOOL* State);
```

#### Type

TCPA protected capability; there must be some evidence of physical access present for the TPM to verify.

#### Parameters

Type	Name	Description
BOOL	State	State to set deactivated flag to

#### Action

The TPM while executing this command MUST obtain assurance from a physical method that operation of this command is authorized.

The TPM SHALL set the TCPA\_PERSISTENT\_FLAGS.deactivated flag to the value in the state parameter.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_BAD_PARAMETER	Parameter was not recognized
TCPA_FAIL	A critical system error occurred

### 8.14.2 TPM\_SetTempDeactivated

#### IDL Definition

```
TCPA_RESULT TPM_SetTempDeactivated();
```

#### Type

TCPA protected capability.

#### Parameters

None.

#### Action

The TPM SHALL set the TCPA\_VOLATILE\_FLAGS.deactivated flag to the value TRUE.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_FAIL	A critical system error occurred

## 8.15 TPM\_FieldUpgrade

### ***Start of informative comment:***

The TPM needs a mechanism to allow for updating the protected capabilities once a TPM is in the field. Given the varied nature of TPM implementations there will be numerous methods of performing an upgrade of the protected capabilities. This command, when implemented, provides a manufacturer specific method of performing the upgrade.

The manufacturer can determine, within the listed requirements, how to implement this command. The command may be more than one command and actually a series of commands.

The IDL definition is to create an ordinal for the command, however the remaining parameters are manufacturer specific.

### ***End of informative comment.***

### **IDL Definition**

```
TCPA_RESULT TPM_FieldUpgrade(
    [in, out] TCPA_AUTH* ownerAuth,
    ... ) ;
```

### **Type**

TCPA protected capability; the TPM Owner must authenticate the command.

### **Parameters**

Type	Name	Description
TCPA_AUTH	ownerAuth	Authentication from TPM owner to execute command
...		Remaining parameters are manufacturer specific

### **Actions**

The TPM SHALL perform the following when executing the command:

- Validate the TPM Owners authorization to execute the command
- Validate that the upgrade information was sent by the TPME. The validation mechanism MUST use a strength of function that is at least the same strength of function as a digital signature performed using a 2048 bit RSA key.
- Validate that the upgrade target is the appropriate TPM model and version.
- Process the upgrade information and update the protected capabilities
- Set the TCPA\_PERSISTENT\_FLAGS.revMajor and TCPA\_PERSISTENT\_FLAGS.revMinor to the values indicated in the upgrade. The selection of the value is a manufacturer option. The values MUST be monotonically increasing. Installing an upgrade with a major and minor revision that is less than currently installed in the TPM is a valid operation.
- Set the TCPA\_VOLATILE\_FLAGS.deactivated to TRUE.

### **Descriptions**

The upgrade mechanisms in the TPM MUST not require the TPM to hold a global secret. The definition of global secret is a secret value shared by more than one TPM.

The TPME is not allowed to pre-store or use unique identifiers in the TPM for the purpose of field upgrade. The TPM MUST NOT use the endorsement key for identification or encryption in the upgrade process. The upgrade process MAY use a TPM Identity to deliver upgrade information to specific TPM's.

The upgrade process can only change protected capabilities.

The upgrade process can only access data in shielded locations where this data is necessary to validate the TPM Owner, validate the TPME and manipulate the blob

The TPM MUST be conformant to the TCPA specification, protection profiles and security targets after the upgrade. The upgrade MAY NOT decrease the security values from the original security target.

The security target used to evaluate this TPM MUST include this command in the TOE.

## 8.16 TPM Internal RSA Operations on Arbitrarily Sized Data

***Start of informative comment:***

The TPM has need to encrypt data either to protect private keys or for other purposes. This function provides a mechanism to perform this encryption in an interoperable way. Typically, this will be required during protected storage operations.

There are two portions of the wrapping process, the first is to “chunk” the area up and the second is to properly encrypt the chunks.

The chunking algorithm allows for different modulus sizes to create the same type of chunks so that the decryption process can properly put the key back together.

The basic format of the key comes from PKCS#1 version 2.0.

***End of informative comment.***



### 8.16.1 TPM\_Internal\_Encrypt

**Start of informative comment:**

This encryption function is for internal TPM use, only. This function takes a private key performs OAEP encoding and additional processing of the encoded blob before performing the actual encryption of the blob.

If an instantiation of security functions contains a TPM, this definition does not prevent the export of an encryption function by that instantiation. That exported encryption function could have the same parameters and data structures as TPM\_Internal\_Encrypt, but must have a different name. It would not be a TCPA protected capability and would not have access to shielded-locations.

Such an encryption function may, or may not, affect the export and import of TCPA compliant equipment to and from sovereign states.

**End of informative comment.**

The definition of this command is for internal use of TPM devices. The TPM MUST NOT export this command outside the TPM.

#### IDL Definition

```
TCPA_RESULT TPM_Internal_Encrypt (
    [in] UINT32 maxPrivWrapSize,
    [in] TCPA_KEY_SLOT target
    [in] TCPA_KEY_SLOT wrapper,
    [in, out] UINT32* WrapSize,
    [out, size_is(*WrapSize)] BYTE* Wrap);
```

#### Type

Internal TCPA protected capability; TPM must not export this command.

#### Parameters

Type	Name	Description
UINT32	maxPrivWrapSize	The maximum size of the output wrapped blob
TCPA_KEY_SLOT	target	This SHALL point to the private key to wrap
TCPA_KEY_SLOT	wrapper	This SHALL be the key that will perform the wrap
UINT32*	WrapSize	The size of the wrapped blob
BYTE*	Wrap	The wrapped blob

#### Action

The TPM SHALL use the RSAES\_OAEP protocol from PKCS#1 version 2.0.

The TPM SHALL create a TCPA\_STORE\_ASYMKEY structure using the information for the key pointed to by target.

After encoding the TCPA\_STORE\_ASYMKEY structure the TPM SHALL fill in the TCPA\_INTERNAL\_HDR structure.

The TPM then creates the blob to be encrypted by appending the TCPA\_INTERNAL\_HDR structure with the TCPA\_STORE\_ASYMKEY structure.

The TPM then encrypts the appended blob using the key pointed to by wrapper.

The TPM returns the wrapped key in the wrap parameter.

### Chunk Calculation

For various encryption algorithms, the size of the key may be longer than a single encryption operation can handle. The following routine provides a standard method of breaking the area into suitable size areas and allowing for the later decryption and re-assembly of the key.

The specification calls for OAEP so this chunk calculation works for the OAEP encryption and encoding method.

- Set *hLen* to 20 bytes (the size of a SHA1 hash)
- Set *int* to the size of the header area TCPA\_INTERNAL\_HDR structure (1 byte)
- Set *k* to the modulus size of the RSA key
- For example for a 2048 bit key, *k* equals 256 bytes
- Formula is  $msize = k - int - 2 * hLen$
- $msize = k - 1 - 2 * hLen = 256 - 1 - 40 = 215$
- Create chunks to encrypt by taking *msize* chunks from the structure and performing the normal OAEP encoding.
- The last chunk does not need to be padded as the process that will recreate the chunks knows the size due to the *dataSize* parameter.

The output area is the wrapped chunks in order of their encryption. It is the responsibility of the receiver of the wrapped area to ensure that the chunks remain in the correct order.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_BAD_PARAMETER	Parameter was not recognized
TCPA_FAIL	A critical system error occurred

## 8.16.2 TPM\_Internal\_Signature

### ***Start of informative comment:***

This signature function is for internal TPM use, only.

If an instantiation of security functions contains a TPM, this definition does not prevent the export of a signature function by that instantiation. That exported signature function could have the same parameters and data structures as TPM\_Internal\_Signature, but must have a different name. It would not be a TCPA protected capability and would not have access to shielded-locations.

### ***End of informative comment.***

The definition of this command is for internal use of TPM devices. The TPM MUST NOT export this command outside the TPM.

### **IDL Definition**

```
TCPA_RESULT TPM_Internal_Signature (
    [in] UINT32 maxSigSize,
    [in] UINT32 blobSize,
    [in] TCPA_PRIVKEY sigKey,
    [in, size_is(blobSize)] BYTE* blob,
    [in, out] UINT32* sigSize,
    [out, size_is(*sigSize)] BYTE* sig);
```

### **Type**

Internal TCPA protected capability; TPM MUST NOT export this command.

### **Parameters**

Type	Name	Description
UINT32	maxSigSize	The maximum size of the outputted wrapped private key
UINT32	blobSize	This SHALL be the size of blob parameter
TCPA_PRIVKEY	sigKey	This SHALL be key that will perform the signature
BYTE*	blob	This SHALL be the data that is to be signed
UINT32*	sigSize	This SHALL be the size of the sig parameter on output
BYTE*	sig	This SHALL be the signature of the blob parameter

### **Action**

The blob MUST be hashed using the SHA-1 algorithm and the resulting TCPA\_DIGEST area is the value to be signed.

The TPM SHALL sign the TCPA\_DIGEST of the blob parameter with the key in sigKey using the RSASSA-PKCS1-v1\_5 protocol from PKCS#1 version 2.0.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_BAD_PARAMETER	Parameter was not recognized
TCPA_FAIL	A critical system error occurred

## 8.17 TPM\_SetRedirection

### **Informative comment**

‘Redirected’ keys enable the output of a TPM to be directed to non-TCPA security functions in the platform, without exposing that output to non-security functions.

It is sometimes desirable to direct the TPM's output directly to specific platform functions without exposing that output to other platform functions. To enable this, the key in a leaf node of TCPA Protected Storage can be tagged as a “redirect” key. Any plaintext output data secured by a redirected key is passed by the TPM directly to specific platform functions and is not interpreted by the TPM.

Since redirection can only affect leaf keys, redirection applies to: TPM\_Unbind, TPM\_Unseal, TPM\_Quote, TPM\_Sign, TPM\_CertifyKey

### **End of informative comments**

### **IDL Definition**

```
TCPA_RESULT TPM_SetRedirection (
    [in, out] TCPA_AUTH* keySlotAuth,
    [AUTH, in] TCPA_KEY_SLOT keySlot,
    [AUTH, in] UINT32 c1,
    [AUTH, in] UINT32 c2);
```

### **Type**

TCPA protected capability; the TPM MAY implement this command. The user MUST supply authorization to use the key pointed to by keySlot.

### **Parameters**

Type	Name	Description
TCPA_AUTH*	keySlotAuth	This SHALL be the authorization to use the key pointed to by keySlot
TCPA_KEY_SLOT	keySlot	This SHALL be slot identifier of a properly loaded key
UINT32	c1	This SHALL be a manufacturer option to specify the output redirection
UINT32	c2	This SHALL be a manufacturer option to provide options for the c1 parameter

### **Action**

The TPM SHALL validate the authorization to use the key pointed to by keySlot.

The TPM SHALL verify that the key pointed to by keySlot has the redirection flag set to TRUE. If FALSE the TPM SHALL return TCPA\_FAIL.

The TPM SHALL set the key slot redirection parameters according to the values in parameters c1 and c2.

A key that is tagged as a “redirect” key MUST be a leaf key in the TCPA Protected Storage blob hierarchy. A key that is tagged as a “redirect” key CAN NEVER be a parent key.

Output data that is the result of a cryptographic operation using the private portion of a “redirect” key:

1. MUST be passed to an alternate output channel
2. MUST NOT be passed to the normal output channel
3. MUST NOT be interpreted by the TPM.

The authorization response returns to the caller.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully
TCPA_BAD_PARAMETER	Parameter was not recognized
TCPA_FAIL	A critical system error occurred

## 9. Subsystem Credentials

### 9.1 Introduction

***Start of informative comment:***

This section defines the credentials by which various entities vouch for a Trusted Platform, plus the Subsystem capabilities that are used during the creation of those credentials.

***End of informative comment.***

All credentials MUST use the TCPA\_VERSION structure.

### 9.2 Endorsement

***Start of informative comment:***

A TPM only has one asymmetric endorsement key pair. Due to the nature of this key pair, both the public and private parts of the key have privacy and security concerns.

Exporting the PRIVEK from the TPM must not occur. This is for security reasons. The PRIVEK is a decryption key and never performs any signature operations.

Exporting the public PUBEK from the TPM under controlled circumstances is allowable. Access to the PUBEK must be restricted to entities that have a “need to know.” This is for privacy reasons.

The PUBEK is tagged with TCPA\_version to indicate the version of the capability that created the key at the time that the key was generated. This may be useful in the event that capabilities are field-upgraded.

Repeated access to the PUBEK of a TPM is desirable in the process of manufacturing TPMs and platforms. Unfortunately, repeated access to the PUBEK is a security concern (because the PUBEK is used to acquire ownership of the TPM) and may be a privacy concern.

The first call to TPM\_CreateEndorsementKeyPair generates the endorsement key pair. After a successful completion of TPM\_CreateEndorsementKeyPair all subsequent calls return TCPA\_FAIL.

The TPM\_ReadPubek returns the PUBEK only while the readPubek flag is TRUE. The owner can set the readPubek flag with an owner authorized command. In order to increase confidence that the PUBEK returned is in response to the command a simple challenge/response is built into the call to TPM\_ReadPubek. The command returns a hash of a submitted nonce and the PUBEK.

***End of informative comment.***

The PRIVEK and PUBEK MUST be accessed only by protected capabilities whose definition explicitly requires access to those keys.

### 9.2.1 TPM\_CreateEndorsementKeyPair

#### IDL Definition

```
TCPA_RESULT TPM_CreateEndorsementKeyPair(
    [in] TCPA_NONCE Nonce,
    [in, size_is(keySize)] TCPA_KEY* keyInfo,
    [out] TCPA_DIGEST* Checksum,
    [out] TCPA_PUBKEY* PubEndorsementKey,
    [out] TCPA_VERSION* Ver);
```

#### Type

TCPA protected capability

#### Parameters

Type	Name	Description
UINT32	Nonce	This is arbitrary data chosen by the entity that submits the command
TCPA_KEY*	keyInfo	The input structure contains all parameters except pubkey and privkey (which are NULL), to specify the size and type of the new key.
TCPA_DIGEST	Checksum	This SHALL be the result of a hash process applied to the concatenation of the PUBEK and the nonce.
TCPA_PUBKEY	PubEndorsementKey	This SHALL be the PUBEK
TCPA_VERSION	Ver	This SHALL be the version specified in section 4.5.

#### Description

For reasons of interoperability, algorithm SHOULD indicate RSA and algParms SHOULD indicate 2048bit endorsement keys. (Refer to Conformance section 10.4 for further details.)

Type	Name	Description
TCPA_PRIVKEY	PRIVEK	This SHALL be the private key of the endorsement key pair.
TCPA_PUBKEY	PUBEK	This SHALL be the public key of the endorsement key pair.

The PRIVEK SHALL exist only in a TCPA-shielded location.

If the data structure TPM\_ENDORSEMENT\_CREDENTIAL is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.

#### Actions

The first valid TPM\_CreateEndorsementKeyPair command received by a TPM SHALL

1. Create a key pair called the “endorsement key pair” using a TCPA-protected capability. The type and size of key are that indicated by algorithm and the algParms.
2. Create “checksum” by appending the nonce to the PUBEK and passing the concatenated data through a hash process.
3. Store the PRIVEK.
4. Export the data structures PUBEK, checksum and TCPA\_version.

Subsequent calls to TPM\_CreateEndorsementKeyPair SHALL return code TCPA\_FAIL.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_BAD_PARAMETER	Parameter not recognized.
TCPA_FAIL	A critical system error occurred.



## 9.2.2 TPM\_ReadPubek

### IDL Definition

```
TCPA_RESULT TPM_ReadPubek(
    [in] TCPA_NONCE Nonce,
    [out] TCPA_DIGEST* Checksum,
    [out] TCPA_PUBKEY* PubEndorsementKey,
    [out] TCPA_VERSION* Ver);
```

### Type

TCPA protected capability

### Parameters

Type	Name	Description
UINT32	Nonce	This is arbitrary data chosen by the entity that submits the command
TCPA_DIGEST	Checksum	This SHALL be the result of a hash process applied to the concatenation of the PUBEK and the nonce.
TCPA_PUBKEY	PubEndorsementKey	This SHALL be the PUBEK
TCPA_VERSION	Ver	This SHALL be the version specified in section 4.5.

### Description

This command returns the PUBEK.

### Actions

The TPM\_ReadPubek command SHALL

1. If readPubek is FALSE return TCPA\_DISABLED\_CMD.
2. Create "checksum" by appending the nonce to the PUBEK and passing the concatenated data through a hash process.
3. Export the PUBEK, checksum and TCPA\_version.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_BAD_PARAMETER	Parameter not recognized. Output areas have a NULL pointer.
TCPA_FAIL	A critical system error occurred.
TCPA_DISABLED_CMD	A previous TPM_DisablePubekRead command has been successfully processed setting TCPA_PERSISTENT_FLAGS.readPubek to FALSE.

### 9.2.3 TPM\_DisablePubekRead

**Start of informative comment:**

The TPM Owner may wish to prevent any entity from reading the PUBEK. This command sets the non-volatile flag so that the read command always returns TCPA\_DISABLED\_CMD.

**End of informative comment.****IDL Definition**

```
TPM_DisablePubekRead(  
    [in, out] TCPA_AUTH* ownerAuth,  
);
```

**Type**

TCPA protected capability; the user must present authorization from the TPM Owner.

**Parameters**

Type	Name	Description
TCPA_AUTH*	ownerAuth	This SHALL be the authorization from the TPM Owner to execute this command

**Actions**

This capability sets the TCPA\_PERSISTENTFLAGS.readPubek flag to FALSE.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_FAIL	A critical system error occurred.

## 9.2.4 TPM\_OwnerReadPubek

### IDL Definition

```
TCPA_RESULT TPM_OwnerReadPubek(
    [in, out] T CPA_AUTH* ownerAuth,
    [AUTH, out] T CPA_PUBKEY* PubEndorsementKey,
    [AUTH, out] T CPA_VERSION* Ver);
```

### Type

TCPA protected capability; caller must supply authorization from the TPM Owner

### Parameters

Type	Name	Description
TCPA_AUTH*	ownerAuth	This SHALL be the authorization from the TPM Owner to execute this command
TCPA_PUBKEY	PubEndorsementKey	This SHALL be the PUBEK
TCPA_VERSION	Ver	This SHALL be the version specified in section 4.5.

### Description

This command returns the PUBEK.

### Actions

The TPM\_ReadPubek command SHALL

1. Validate the TPM Owner authorization to execute this command
2. Export the PUBEK and T CPA\_version.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_BAD_PARAMETER	Parameter not recognized.
TCPA_FAIL	A critical system error occurred.
TCPA_AUTHFAIL	The Owner authorization did not pass

### 9.3 Generating a Trusted Platform Module Identity

***Start of informative comment:***

The purpose of TPM\_MakeIdentity is to create

- an asymmetric key pair within the Trusted Platform Module and
- evidence that the key pair is bound to a label.

Only the Owner of the TPM has the privilege of creating a TPM identity. (An identity is not activated until the reception of the command TPM\_ActivateIdentity.)

TPM\_MakeIdentity communicates new authorization data to the TPM using almost the same process as Protected Storage uses to communicate new authorization data for blobs. Both processes require the creation of a TPM\_OSAP session and the use of the session's shared secret to XOR the new authorization data. The requirement for TPM\_MakeIdentity is that the TPM\_OSAP session must start with the TPM Owner authorization.

The authorization data will provide the ability to associate authorization sessions with the new identity in the future. The protection of the authorization data comes from the XOR having a one-time pad nature to it. If an attacker can determine the shared secret of the TPM\_OSAP session then the attacker can learn the new value of the authorization data. For the case of identities, the owner is always the SRK, which in many cases has well-known authorization data. This would allow an attacker to determine what the shared secret was and hence what the value of the new authorization data is.

To avoid the problem with the SRK, the TPM\_MakeIdentity command requires the TPM\_OSAP session to use the TPM Owner as the authorization to establish the session. This creates a shared secret that only the TPM Owner and the TPM know and allows the proper protections when using the XOR for encryption.

A tpm\_signature\_key must be known only to the TPM.

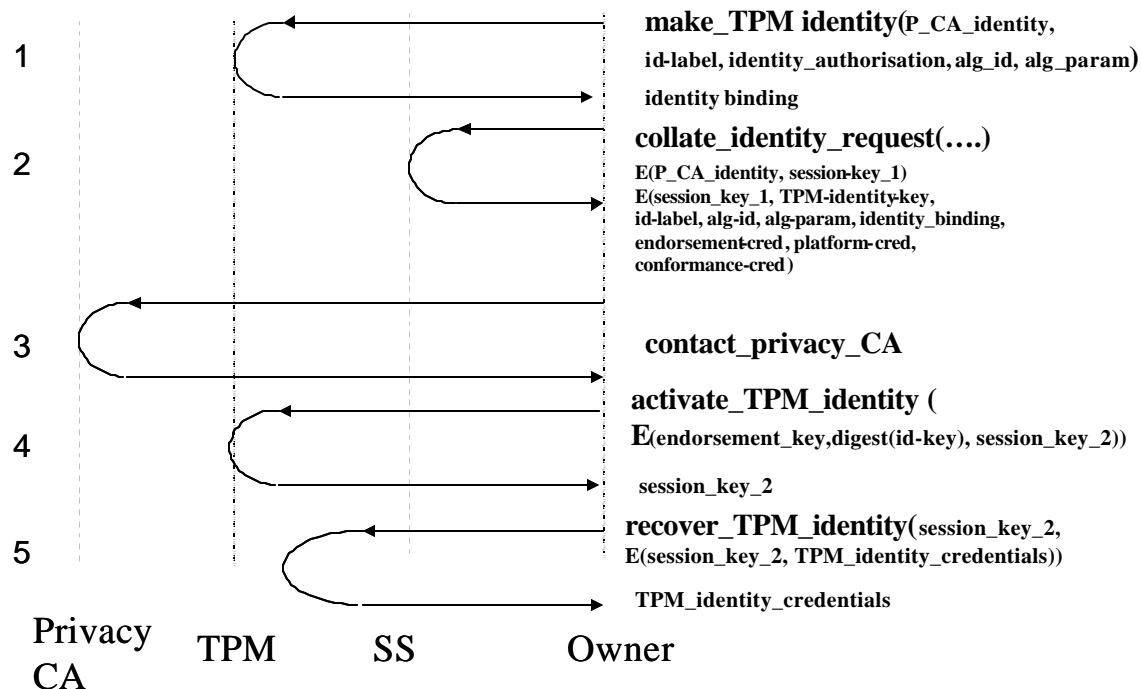
Identity\_binding uses the private (signature) key of a TPM identity. The private (signature) key of a TPM identity is available only to selected commands. Its use enables a recipient to be certain that identity\_binding was generated inside a TPM. This feature prevents a rogue Owner from assembling identity\_binding data structures outside the TPM and hence obtaining attestation to the same TPM identity from multiple Privacy CAs.

Identity\_binding is tagged with TCPA\_version so as to indicate the version of the capability that created the identity\_binding at the time that identity\_binding was generated. This may be useful in the event that capabilities are field-upgraded.

The algorithm parameter indicates the type of encryption algorithm in use for the TPM identity. It may indicate RSA, or ECC, to give two examples. The algParms references the parameters that are necessary for the particular encryption algorithm in use. For RSA, these parameters are just the length of the RSA key.

The PKI identity protocol enables a Trusted Platform Module to have multiple identities. Each identity may have attestation from exactly one Privacy CA.

## Obtaining a TPM identity



The TPM creates an identity-binding signature (the value of a signature over the TCPA\_IDENTITY\_CONTENTS structure). Among other things, this proves possession of the new private key, which does the signing of the TCPA\_IDENTITY\_CONTENTS structure. The Subsystem sends the signature along with evidence of a genuine TPM and the platform the TPM resides on to a Privacy CA. The encryption of the request is to provide privacy not security.

The Privacy CA inspects the evidence and concurs that the TPM is genuine and in a valid platform. The Privacy CA validates the signature of the TCPA\_IDENTITY\_CONTENTS structure and verifies that it was signed using the private key corresponding to the public key in the identity request. The TCPA\_IDENTITY\_CONTENTS structure includes the Privacy CA's public key. The Privacy CA obtains assurance that it (and not some other Privacy CA) is the target of the request to provide the identity attestation.

The Privacy CA cannot check that the public key inside identity-binding signature belongs to a genuine TPM, but it knows that the TPM described in the evidence is a genuine TPM. The Privacy CA generates the attestation credential and encrypts the credential for decryption by the requesting TPM. The Privacy CA also sends the genuine TPM a "statement" that the credential attests to a particular public key (the one in the identity-credential).

The TPM receives the encrypted data. It cannot parse the credential, but it can check that the credential attests to one of its public keys, by checking the "statement" from the Privacy CA. Only if the credential relates to one of the TPM's public keys does the TPM enable recovery of the credential.

The presumption is that the Privacy CA is trustworthy. This must be the case for the acceptance of the attestation by a third party. Hence, if the attestation is worth having, the "statement" from the Privacy CA to the TPM can be trusted. Hence, the TPM "knows" that the encrypted credential relates to the public key in the "statement." The Privacy CA has ensured that only a genuine TPM can recover the encrypted credential and statement and that a genuine TPM will enable recovery of the credential only if the credential is associated with a public key belonging to the TPM.

A rogue can certainly pose as a Privacy CA and cause the TPM to release the credential created by that rogue. But who will trust the attestation provided by that rogue? A trustworthy credential can be recovered only if it attests to a public key of a genuine TPM, because the Privacy CA that created the credential can

be trusted to check that a TPM is genuine and to correctly state that a credential describes a particular public key, and a genuine TPM checks that the public key belongs to that TPM before releasing the credential.

The reason for including the public key of the Privacy CA inside identity-binding signature is to prevent a rogue obtaining attestation from multiple Privacy CAs. The identity-binding signature creation is an atomic operation performed at the same time as the key pair creation, and therefore the TPM cannot be coerced into creating a version of the identity-binding signature with the same keys but a different Privacy CA public key.

The Identity-binding signature is one of the few operations that are permitted to use the private (signature) key of a TPM identity. A version of identity\_binding with a different Privacy CA public key can't be reproduced by commands from outside the TPM, because the TPM will refuse to sign arbitrary data with a private (signature) key of a TPM identity.

The process deliberately has certain characteristics:

For example, during TPM\_MakeIdentity,

- The atomic generation of the key pair and encrypted identity\_binding information prevents the creation by a TPM of duplicate identity\_binding information while avoiding the need for a TPM to retain state.
- Signing with the private (signature) key of a TPM identity prevents the creation of duplicate "identity\_binding" information outside a TPM.
- When a Privacy CA receives data, it can use the data describing the new TPM identity to check that the request for attestation (if it came from a genuine TPM) is a unique request, use the endorsement credentials to check that a stated TPM is a genuine TPM, and use the platform credentials and conformance credentials to check that a stated platform is a genuine Trusted Platform. The Privacy CA *cannot*, however, verify that the new TPM identity was actually generated by that genuine TPM. On the assumption, however, that the new TPM identity was actually generated by a genuine TPM, the Privacy CA generates TPM\_IDENTITY\_CREDENTIALs and a statement that expresses a binding between that TPM\_IDENTITY\_CREDENTIAL and the new TPM identity. The Privacy CA then encrypts this information so that it can be recovered only by the genuine TPM described by the endorsement credentials.
- During TPM\_ActivateIdentity, the genuine TPM checks that the encrypted TPM\_IDENTITY\_CREDENTIAL is bound to one of the TPM's identities and enables decryption of TPM\_IDENTITY\_CREDENTIAL *only* if that association exists. This last stage is critical but subtle, since the TPM has insufficient computing power to parse TPM\_IDENTITY\_CREDENTIAL and relies on the "statement" from the Privacy CA that a TPM\_IDENTITY\_CREDENTIAL is associated with a given identity.
- The entire process depends critically on the trustworthiness of the Privacy CA. If the Privacy CA is trustworthy, a plaintext TPM\_IDENTITY\_CREDENTIAL recovered by a TPM describes an identity of a genuine TPM. Otherwise, a TPM\_IDENTITY\_CREDENTIAL cannot be trusted. The Privacy CA must be trusted to make TPM\_IDENTITY\_CREDENTIAL only if the request for attestation is a unique request and the stated TPM and platform are genuine. The Privacy CA must be trusted never to reveal a plaintext copy of TPM\_IDENTITY\_CREDENTIAL and to be truthful when stating that a particular TPM\_IDENTITY\_CREDENTIAL is associated with a particular identity.

***End of informative comment.***

### 9.3.1 TPM\_MakeIdentity

#### IDL Definition

```

TCPA_RESULT TPM_MakeIdentity(
    [in, out] TCPA_AUTH* TpmOwnerAuth,
    [in, out] TCPA_AUTH* SrkAuth,
    [AUTH, in] TCPA_PUBKEY PrivacyCA,
    [AUTH, in] UINT32 LabelSize,
    [AUTH, in] UINT32 MaxWrapSize,
    [AUTH, in] UINT32 MaxCASize,
    [AUTH, in] UINT32 IdentityAuthSize,
    [AUTH, in, size_is(LabelSize)] BYTE* Label,
    [AUTH, in] TCPA_KEY keyInfo,
    [AUTH, in, size_is(IdentityAuthSize)] BYTE* IdentityAuth,
    [AUTH, in, out] UINT32* WrapSize,
    [AUTH, in, out] UINT32* CaSize,
    [AUTH, out] TCPA_PUBKEY* IdentityPub,
    [AUTH, out, size_is(*WrapSize)] BYTE* Wrap,
    [AUTH, out, size_is(*CaSize)] BYTE* identityBinding); )

```

#### Type

TCPA protected capability; user must provide authorizations from the TPM Owner and the SRK.

#### Parameters

Type	Name	Description
TCPA_AUTH	TpmOwnerAuth	This SHALL be authorization from the TPM Owner. The authorization session MUST be OSAP.
TCPA_AUTH	SrkAuth	This SHALL be authorization for use of the SRK
TCPA_PUBKEY	PrivacyCA	This SHALL be the Public key of the Privacy CA that will vouch for this TPM identity.
UINT32	LabelSize	The size of the label field
UINT32	MaxWrapSize	The maximum size of the wrap area
UINT32	MaxCASize	The maximum size of the CA area
UINT32	IdentityAuthSize	The size of the area for the new identities authorization
BYTE*	Label	The label for the new identity
TCPA_KEY	keyInfo	The input structure contains all parameters except pubkey and privkey (which are NULL), to specify the size and type of the new key.
BYTE*	IdentityAuth	This SHALL be an encrypted version of the authorization data (identity_authorization) that is to be presented when accessing the private key of this new TPM identity.
UINT32*	WrapSize	The size of the outputted wrapped identity.
UINT32*	CaSize	The size of the area to be sent to the CA
TCPA_PUBKEY*	IdentityPub	This SHALL be the public key of this TPM identity.
BYTE*	Wrap	This SHALL be a protected-storage structure, used to store the private key of this TPM identity.

BYTE*	identityBinding	This SHALL be the signature value of the signature over the structure TCPA_IDENTITY_CONTENTS, using the tpm_signature_key in the algorithm indicated by asym_alg_id and asym_alg_parameters.
-------	-----------------	--

### Description

The command TPM\_MakeIdentity is used to generate an identity in a TPM and to request attestation to that identity.

The public key of the new TPM identity SHALL be identityPubKey. The private key of the new TPM identity SHALL be tpm\_signature\_key.

For reasons of interoperability, algorithm SHOULD indicate RSA and algParms SHOULD indicate a 2048bit TPM identity key. (Refer to Conformance section 10.4.1 for further details.)

### Properties of the new identity

Type	Name	Description
TCPA_PUBKEY	IdentityPubKey	This SHALL be the public key of a previously unused asymmetric key pair.
TCPA_PRIVKEY	Tpm_signature_key	This SHALL be the private key that forms a pair with identityPubKey and SHALL be extant only in a TCPA-shielded location.

This capability also generates a blob containing the tpm\_signature\_key, which has the same format as a blob created by a CreateWrapKey command. The structure of that blob is defined in TCPA\_PRIVKEY.

If identityPubKey is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.

If the Subsystem supports the creation of an audit log, the receipt-event and response-event associated with a TPM\_MakeIdentity command SHALL form part of that log.

### Actions

A Trusted Platform Module that receives a valid TPM\_MakeIdentity command SHALL do the following:

1. Use TpmOwnerAuth to verify that the Owner authorized all TPM\_MakeIdentity parameters tagged with "AUTH IN", and abandon this TPM\_MakeIdentity process if there is no match. The TPM MUST use a protected capability to verify the authorization data.
2. Use SRK\_auth to verify that the SRK owner authorised all TPM\_MakeIdentity parameters tagged with "AUTH IN", and abandon this TPM\_MakeIdentity process if there is no match. The TPM MUST use a protected capability to verify the authorization data.
3. Obtain the identity\_authorization to be associated with the new TPM identity, by decrypting the field IdentityAuth using the shared secret created with the TPM\_OSAF session. The establishment of the TPM\_OSAF session MUST use the authentication of the TPM Owner. The decryption uses an XOR of the identityAuth field and the shared secret of the TPM\_OSAF session.
4. Create an asymmetric key pair (identityPubKey and tpm\_signature\_key) using a TCPA-protected capability, in accordance with the algorithm and algParms contained in the TPM\_MakeIdentity command
5. Associate the tpm\_signature\_key with the identity\_authorization, such that tpm\_signature\_key can only be used upon presentation of identity\_authorization.



6. Export a data structure with the same format as that created by the command "CreateWrapKey," using identity\_authorization as authorization data, using the Storage Root Key as the parent key, and marking tpm\_signature\_key as belonging to a TPM identity.
7. Export the data structure identityPubKey using an authorization method that identifies identityPubKey as a response to this instance of a TPM\_MakeIdentity command. The authorization method SHALL be a TCPA-protected capability whose purpose is to provide such authorization data.
8. Export the data structure identity\_binding.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_BAD_PARAMETER	Parameter was not recognized.
TCPA_FAIL	A critical system error occurred.

### 9.3.2 TSS\_CollateIdentityRequest

**Start of informative comment:**

The purpose of the TSS\_CollateIdentityRequest command is to assemble all the data that will be required by a Privacy CA in order to assess a platform and attest to the identity of a Subsystem.

The TSS\_CollateIdentityRequest command is separate from the TPM\_MakeIdentity command because their processing might be done on different engines. The reason is that TSS\_CollateIdentityRequest does not have to be trustworthy but TPM\_MakeIdentity must be trustworthy. Therefore, an implementation of TSS\_CollateIdentityRequest does not require the same protection as an implementation of TPM\_MakeIdentity.

A session key (the random number) is used to provide confidentiality of the "TCPA\_IDENTITY\_REQ." This is to ensure that only the Privacy CA chosen by the Owner can interpret the data, while minimizing exposure of that Privacy CA's identity (public) key.

The data asym\_alg\_id indicates the type of encryption algorithm that is used by the TPM identity. To give two examples, it may indicate RSA, or ECC. The data asym\_alg\_parameters references parameters that are necessary for the particular encryption algorithm in use. For RSA, these parameters are just the length of the RSA key.

Once the data structure TCPA\_IDENTITY\_REQ has been produced, it should be sent to the Privacy CA chosen by the Owner.

**End of informative comment.**

#### IDL Definition

```
TCPA_RESULT TSS_CollateIdentityRequest(
    [in] TCPA_VERSION Ver,
    [in] UINT32 Algorithm,
    [in] UINT32 LabelSize,
    [in] UINT32 ParmSize,
    [in] UINT32 EndorseSize,
    [in] UINT32 PlatformSize,
    [in] UINT32 ConformSize,
    [in] UINT32 MaxReqSize,
    [in] UINT32 IdentityBindingSize,
    [in] TCPA_PUBKEY CaPubKey,
    [in] TCPA_PUBKEY IdentityKey,
    [in, out] UINT32* ReqSize,
    [in, size_is(LabelSize)] BYTE* LabelArea,
    [in, size_is(ParmSize)] BYTE* AlgParms,
    [in, size_is(EndorseSize)] BYTE* EndorseCredential,
    [in, size_is(PlatformSize)] BYTE* PlatformCredential,
    [in, size_is(ConformSize)] BYTE* ConformCredential,
    [in, size_is(IdentityBindingSize)] BYTE* IdentityBinding,
    [out, size_is(*ReqSize)] BYTE* IdentityReq);}
```

#### Type

TSS capability and MAY be TPM capability.

#### Parameters

Type	Name	Description
TCPA_VERSION	Ver	This SHALL be the version specified in section 4.5.
UINT32	Algorithm	This SHALL be the type of symmetric encryption algorithm to be used for a session key.

		algorithm to be used for a session key
UINT32	LabelSize	This SHALL be the size of the identity label
UINT32	ParmSize	This SHALL be the size of the symmetric parameters
UINT32	EndorseSize	This SHALL be the size of the endorsement credential
UINT32	PlatformSize	This SHALL be the size of the platform credential
UINT32	ConformSize	This SHALL be the size of the conformance credential
UINT32	MaxReqSize	This SHALL be the maximum size of the output request area.
UINT32	IdentityBindingSize	This SHALL be the identity structure from the TPM_MakeIdentity function.
TCPA_PUBKEY	CaPubKey	This SHALL be the identity (public) key of the entity (Privacy CA) chosen by the Owner to attest to the identity of the Subsystem.
TCPA_PUBKEY	IdentityKey	This SHALL be the public key of the TPM identity for which attestation is requested.
UINT32*	ReqSize	This SHALL be the size of the identityReq field
BYTE*	LabelArea	This SHALL be the identity label
BYTE*	AlgParms	This SHALL be a structure particular to type of symmetric encryption algorithm to be used for a session key
BYTE*	EndorseCredential	This SHALL be a TCPA-defined data structure which contains the data of TPM_ENDORSEMENT_CREDENTIAL and attests that a specific TPM conforms to the TCPA specification.
BYTE*	PlatformCredential	This SHALL be a TCPA-defined data structure which contains the data of platform_credential and attests that a specific platform conforms to the TCPA specification.
BYTE*	ConformanceCredential	This SHALL be a TCPA-defined data structure which contains the data of conformance-credential and attests that the design of a specific platform conforms to the TCPA specification.
BYTE*	IdentityBinding	This SHALL be the data structure exported by the command TPM_MakeIdentity.
BYTE*	IdentityRequest	This SHALL be the data structure defined in this section.

## Description

The command `TSS_CollateIdentityRequest` assembles all data necessary to request attestation of a Trusted Platform Module identity.

A Trusted Platform Subsystem that receives a valid `TSS_CollateIdentityRequest` command SHALL export the data structure “`TCPA_IDENTITY_REQ`.”

The TSS in executing this function performs two encryptions. The first is to symmetrically encrypt the information and the second is to encrypt the symmetric encryption key with an asymmetric algorithm. The symmetric key is a random nonce and the asymmetric key is the public key of the CA that will provide the identity credential.

### Actions

The command SHALL perform the following actions:

1. Validate that the TSS can support the symmetric algorithm and the asymmetric algorithm necessary to perform the encryptions. If the TSS does not support these algorithms it MUST return `TCPA_BAD_PARAMETER`.
2. Initialize the `identityRequest` area to be the `TCPA_IDENTITY_REQ` structure.
3. Create a session key by calling `TSS_GetRandom`.
4. Create an IV for the symmetric encryption. The IV is stored in the `algParms` field of the `TCPA_IDENTITY_REQ` structure.
5. Create the `TCPA_SYM_IDENTITY_REQ` structure. The command SHALL fill in each field of the structure according to the field requirements.
6. Encrypt the `TCPA_SYM_IDENTITY_REQ` structure using the session key and the symmetric algorithm from the algorithm parameter.
7. Place the encrypted `TCPA_SYM_IDENTITY_REQ` blob into the `TCPA_IDENTITY_REQ.symBlob` field.
8. Create a `TCPA_ASYM_IDENTITY_REQ` structure and set `TCPA_ASYM_IDENTITY_REQ.sessionkey` to the session key created in step 3.
9. Encrypt the `TCPA_ASYM_IDENTITY_REQ` structure using the algorithm specified by the type of key in `caPubKey`. The key for the encryption is `caPubKey`.
10. Place the encrypted `TCPA_ASYM_IDENTITY_REQ` blob into the `TCPA_IDENTITY_REQ.asymBlob` field.
11. Return the `TCPA_IDENTITY_REQ` structure.

Return Value	Description
<code>TCPA_SUCCESS</code>	Operation completed successfully.
<code>TCPA_BAD_PARAMETER</code>	Parameter not recognized.
<code>TCPA_FAIL</code>	A critical system error occurred.

### 9.3.3 Contacting a Privacy CA

***Start of informative comment:***

The operations and procedures of a Privacy CA are outside the scope of this specification.

The anticipation, however, is that a Privacy CA will use at least the following checks before agreeing to attest to a TPM identity for a platform:

- Interpret the data structure “TCPA\_IDENTITY\_REQ” in the supplied data and validate the various fields in the structure.
- The verification of the privacy CA’s public is inherent in the decryption of the TCPA\_IDENTITY\_REQ structure. If the decryptions yield valid structures then the key was correct otherwise, the structures are not properly formed and the key was bad.
- Interpret the conformance credential information in the supplied data in order to verify that the design of the platform meets the TCPA specification and is in accordance with the policies of the Privacy CA.
- Interpret the platform-credential information in the supplied data in order to verify that the construction of the platform meets the TCPA specification and is in accordance with the policies of the Privacy CA.
- Interpret the endorsement-credential information in the supplied data in order to verify that the construction of the TPM meets the TCPA specification and is in accordance with the policies of the Privacy CA.
- Create a TCPA\_IDENTITY\_CONTENTS structure and validate the signature of the area provided by the new identity.

It is anticipated that a Privacy CA will then take the following actions:

1. Using the supplied data, construct a TPM-identity-credential according to the TCPA specification, and sign the instantiation using a private key belonging to the Privacy CA.
2. Generate a session key. The assumption is that the session key comes from a suitable random number generator that provides a suitable level of entropy.
3. Create the TCPA\_SYM\_CA\_ATTESTATION structure.
4. Store the session key in TCPA\_ASYM\_CA\_CONTENTS.
5. Create a digest of the identityPubKey. Store the digest value in TCPA\_ASYM\_CA\_CONTENTS.
6. Encrypt the TCPA\_ASYM\_CA\_CONTENTS structure using the PUBEK sent in the attestation request.
7. Return the TCPA\_SYM\_CA\_ATTESTATION structure and the encrypted TCPA\_ASYM\_CA\_CONTENTS structure

The symmetric algorithm should be the same algorithm that the TSS used in creating the TCPA\_IDENTITY\_REQ structure. The asymmetric algorithm must be the algorithm that is defined by the type of PUBEK.

***End of informative comment.***

### 9.3.4 TPM\_ActivateTPMIdentity

**Start of informative comment:**

The purpose of TPM\_ActivateIdentity is to twofold. The first purpose is to obtain assurance that the credential in the TCPA\_SYM\_CA\_ATTESTATION is for this TPM. The second purpose is to obtain the session key used to encrypt the TPM\_IDENTITY\_CREDENTIAL.

TPM\_ActivateIdentity checks that the symmetric session key corresponds to a TPM-identity before releasing that session key.

Only the Owner of the TPM has the privilege of activating a TPM identity. The Owner is required to authorize the TPM\_ActivateIdentity command. The owner may authorize the command using either the TPM\_OIAP or TPM\_OSAP authorization protocols.

**End of informative comment.**

**IDL Definition**

```
TCPA_RESULT TPM_ActivateTPMIdentity(
    [in, out] TCPA_AUTH* TpmOwnerAuth,
    [in, out] TCPA_AUTH* identityAuth,
    [AUTH, in] TCPA_KEY_SLOT Identity,
    [AUTH, in] UINT32 BlobSize,
    [AUTH, in, size_is(BlobSize)] BYTE* Blob,
    [AUTH, out] TCPA_NONCE* SymmetricKey);
```

**Type**

TCPA protected capability; user must provide authorization from the TPM Owner to execute command.

**Parameters**

Type	Name	Description
TCPA_AUTH	TpmOwnerAuth	This SHALL be the authorization from the TPM Owner to execute this command
TCPA_AUTH	identityAuth	This SHALL be the authorization to use the identity key (this is to allow for the authorized execution of the internal validation)
TCPA_KEY_SLOT	Identity	This SHALL be the public key of the TPM identity that is intended to be activated.
UINT32	BlobSize	This SHALL be the size of the blob field
BYTE*	Blob	This SHALL be the encrypted TCPA_ASYM_CA_CONTENTS structure from a privacy CA
TCPA_NONCE	SymmetricKey	This SHALL be the plaintext value of the session key recovered from TCPA_ASYM_CA_CONTENTS structure

**Description**

The command TPM\_ActivateIdentity activates a TPM identity created using the command TPM\_MakeIdentity.

The command assumes the availability of the private key associated with the identity. The command will verify the association between the keys during the process.

The command will decrypt the TCPA\_ASYM\_CA\_CONTENTS structure, extract the session key and verify the connection between the public and private keys.

**Actions**

2. Using the identityAuth field, validate the authorization to execute commands using the identity.
3. Decrypt the TPCA\_CA\_ASYM\_CONTENTS structure. The decryption key is the TPM PRIVEK.
4. Compute a digest of the public key in the identity parameter. Compare the computed digest to the value in the decrypted TPCA\_CA\_ASYM\_CONTENTS structure. Return with the error code TPCA\_BAD\_PARAMETER on a mismatch.
5. Validate that the identity private key belongs to this TPM.
6. Validate that the identity public key is the public key of a valid TPM identity. The validation process does the following:
  - Generate a nonce from the TPM RNG.
  - Encrypt the nonce using the public key pointed to by identity.
  - Decrypt the nonce using the private key assumed to be associated with the identity.
  - Verify that the decryption matches the generated nonce.
7. Return the session key from the TPCA\_CA\_ASYM\_CONTENTS structure.

Return Value	Description
TPCA_SUCCESS	Operation completed successfully.
TPCA_BAD_PARAMETER	Parameter was not recognized.
TPCA_FAIL	A critical system error occurred.

### 9.3.5 TSS\_RecoverTPMIdentity

#### ***Start of informative comment:***

The purpose of TSS\_RecoverIdentity is to recover a plaintext copy of the data structure TPM\_IDENTITY\_CREDENTIAL that attests that a particular identity belongs to a genuine TPCA Trusted Platform.

The TSS\_RecoverIdentity command is separate from the TPM\_ActivateIdentity command because their processing might be done on different engines. The reason is that TSS\_RecoverIdentity does not have to be trustworthy but TPM\_ActivateIdentity must be trustworthy. Therefore, an implementation of TSS\_RecoverIdentity does not require the same protection as an implementation of TPM\_ActivateIdentity.

Exactly one entity may attest to a TPM identity.

Access to the TPM\_IDENTITY\_CREDENTIAL must be restricted to entities that have a “need to know.” This is for reasons of privacy.

#### ***End of informative comment.***

The command TSS\_RecoverIdentity obtains a plaintext copy of the TPM\_IDENTITY\_CREDENTIAL created by a Privacy CA.

If the data structure TPM\_IDENTITY\_CREDENTIAL is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is only available to authorized entities.

#### **IDL Definition**

```
TPCA_RESULT TSS_RecoverTPMIdentity(
```

```

[in] TCPA_NONCE SessionKey,
[in] UINT32 symAttSize,
[in] UINT32 MaxCredentialSize,
[in, size_is(symAttSize)] BYTE* symAtt,
[in, out] UINT32* CredentialSize,
[out, size_is(*CredentialSize)] BYTE* Credential);

```

### Type

This is a TSS capability

### Parameters

Type	Name	Description
TCPA_NONCE	SessionKey	This SHALL be the symmetric key decrypted by the TPM_ActivateIdentity
UINT32	symAttSize	This SHALL be the size of the symAtt parameter
UINT32	MaxCredentialSize	This SHALL be the maximum size of the credential to be output
BYTE*	symAtt	This SHALL be the TCPA_CA_SYM_ATTESTATION structure
UINT32*	CredentialSize	This SHALL be the size of the credential
BYTE*	Credential	This SHALL be the decrypted TCPA_IDENTITY_CREDENTIAL

### Actions

A Trusted Platform Subsystem that receives a valid TSS\_RecoverIdentity command SHALL do the following:

1. Using the session key and the symmetric algorithm indicated by algorithm and the algorithm parameters, decrypt credential parameter inside TCPA\_CA\_SYM\_ATTESTATION to recover the TPM\_IDENTITY\_CREDENTIAL.
2. The TSS SHOULD verify the self-consistency of TPM\_IDENTITY\_CREDENTIAL and abandon this TSS\_RecoverIdentity process if there is an inconsistency.
3. Export TPM\_IDENTITY\_CREDENTIAL.

Return Value	Description
TCPA_SUCCESS	Operation completed successfully.
TCPA_BAD_PARAMETER	Parameter not recognized.
TCPA_FAIL	A critical system error occurred.



e a length of 128 bits for one symmetric cipher, 168 for another, and 256 for yet another, but the TPM does not need to determine this from “length3.” Instead, it simply reads to the end of the string).

In short, therefore, the TPM does the following on decryption:

- skips five bytes;
- reads the next (say 160, if SHA-1 is used) bits and compares this to the table of hashed, inactivated public keys that it has stored(if there is a match it proceeds, otherwise, it aborts the operation);
- skips the next three bytes;
- reads the remaining bytes (until the end of the string) into a buffer; and
- returns this buffer to the Owner as the symmetric key.

***End of informative comment.***

#### **9.4.1 From Owner to Privacy CA**

The protocol from the Owner to the Privacy CA SHALL consist of the following IdentityRequest message:

```
IdentityRequest ::= SEQUENCE {  
    version          Version,  
    encSessionKey    EncSessionKey,  
    encRequest       EncRequest  
}
```

```

Version ::= INTEGER
-- the version number, for compatibility with future revisions of this
specification. It shall be 0 for this version of the specification.

EncSessionKey ::= BIT STRING
-- the ciphertext resulting from the encryption (under the public identity
key of the Privacy CA) of a randomly generated symmetric key (which itself is
DER-encoded as a BIT STRING).

EncRequest ::= BIT STRING
-- the ciphertext resulting from the encryption (under the session key
above) of the following DER-encoded data structure:

"Request ::= SEQUENCE {
  tpmIdKey      SubjectPublicKeyInfo, -- new public key
  tpmIdLabel    OCTET STRING,        -- identity label
  tcpaVersion   TCPASpecVersion,     -- "major.minor"
  identityBinding BIT STRING,        -- (see below)
  endorsementCred Certificate,       -- X.509v3 PK cert
  platformCred  Certificate,         -- X.509 attr. cert
  conformanceCred Certificate        -- X.509 attr. cert
}"

--
-- SubjectPublicKeyInfo
(a SEQUENCE of an AlgorithmIdentifier and a BIT STRING) is specified in
X.509. The BIT STRING contains the subject's public key (for example, if the
algorithm specified is rsaEncryption, the BIT STRING contains the BER
encoding of a value of PKCS #1 type "RSAPublicKey").

-- "identityBinding" is the signature value(using the newly generated TPM
private key that corresponds with the tpmIdKey) over the data specified in
Section. How that data is formatted or delimited is beyond the scope of the
protocol specified here; however, the formatting chosen must be known to
both the TPM and the Privacy CA.

```

#### 9.4.2 From Privacy CA to Owner

The protocol from the Privacy CA to the Owner consists of the PCAResponse message:

```

PCAResponse ::= SEQUENCE {
  version      Version,
  symmAlg      AlgorithmIdentifier,
  encActivationKey EncActivationKey,
  enctpmIdCred EnctpmIdCred
}
EncActivationKey ::= BIT STRING
-- the ciphertext resulting from the encryption (under the PUBEK of the TPM)
of the following DER-encoded data structure:
--
-- ActivationKey ::= SEQUENCE {
--   idKeyDigest    BIT STRING, -- hash of tpmIdKey
--   symmetricKey   BIT STRING
-- }
--
-- NOTE: the validity of the entire protocol for obtaining a TPM identity
depends critically upon the assumption that a genuine TPM will only ever

```

decrypt data using its PRIVEK as part of the TPM\_ActivateIdentity() call. An Owner will never be able to ask a TPM for the decryption of arbitrary data that has been encrypted with its PUBEK. Furthermore, the difficulty of successfully impersonating a TPM is ultimately bound to the computational complexity of finding a collision for idKeyDigest. It is therefore STRONGLY RECOMMENDED that the digest be computed using the full output of a cryptographic hash algorithm of sufficient strength (e.g., the full 160 bits of SHA-1).

EnctpmIdCred ::= BIT STRING

-- the ciphertext resulting from the encryption (under the symmetric activation key above) of the tpmIdentityCred (which is itself DER-encoded as an X.509 PK Certificate).

## 9.5 Instantiation of Credentials as Certificates

***Start of informative comment:***

Unambiguous definition of a data structure containing credentials is necessary if those credentials are to be communicated between platforms. A certificate is such an unambiguous definition.

***End of informative comment.***

### **Certificate syntax**

TCPA certificate syntax conforms with the definitions for public-key certificates and attribute certificates in X.509. The following TCPA certificate types are public-key certificates:

- TPM endorsement certificate
- TPM identity certificate

The following TCPA certificate types are attribute certificates:

- Platform endorsement certificate
- Platform conformance certificate
- Validation data certificate

The form of the following certificates is out of scope for this version of the TPM specification:

- TPM endorsement entity certificate
- TCPA component endorsement entity certificate
- Platform endorsement entity certificate
- Platform conformance certificate

### 9.5.1 Instantiation of TPM\_ENDORSEMENT\_CREDENTIALs

**Start of informative comment:**

An endorsement certificate is an instantiation of an TPM\_ENDORSEMENT\_CREDENTIAL.

Access to an endorsement certificate must be restricted to entities that have a “need to know.” This is for reasons of privacy.

This definition assumes that the PUBEK is a 2048bit RSA keys.

**End of informative comment.**

If the data structure <endorsement\_certificate> is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.

#### Overview

The TPM endorsement certificate represents an assertion by the TPM endorsement entity that the referenced TPM conforms with the TCPA TPM specification.

#### Profile

Notes:

- Some fields are assigned a value even though the certificate user performs no action based on that value. In such cases, the intention is to inhibit non-TCPA implementations from making inappropriate use of the certificate.
- It is intended that the lifetime of a TPM will be shorter than the crypto-period of the TPM endorsement public and private keys. Therefore, keys are not “rolled-over”.
- The trustworthiness of the architecture is vulnerable to the compromise of a single TPM endorsement private key. However, the architecture does not include a revocation mechanism. Nevertheless, certain forms of revocation scheme can be retrofitted, should it become necessary at some time in the future.

In the case of the TPM endorsement certificate, the **issuer** is the TPM endorsement entity and the **user** is a Privacy CA.

Field	Issuer action	User action
Version	Assign value 2 (v3).	Check value = 2, else reject.
Serial number	Assign a value unique amongst all certificates issued by “issuer”.	Use in validating the platform endorsement and conformance certificates.
Signature	Assign the algorithm identifier sha-1WithRSAEncryption (1:2:840:113549:1:1:5).	Check the algorithm identifier = 1:2:840:113549:1:1:5, else reject. Validate the signature on the certificate using the public key of the TPME (which shall be a 2048-bit RSA key), obtained by an out-of-band means and referenced by “issuer” and “authority key identifier”.
Issuer	The distinguished name of the TPM endorsement entity. That is the entity that asserts that the subject TPM conforms with the TCPA specification. (Note: this may be the TPM manufacturer or a conformance test laboratory.)	Check that the name is the name of one of the acceptable TPM endorsement entities, use in validating the platform endorsement and conformance certificates.

Validity	Assign notBefore to the current time and notAfter to a later time (maybe the latest time permitted by the encoding scheme).	Check that the current time is later than the notBefore time, else reject.
Subject	Assign the value NULL.	No action.
Subject public key info	Assign algorithm identifier RSAES-OAEP (1:2:840:113549:1:1:7). Include a 2048-bit RSA public key for key encipherment with OAEP formatting. (Note: this is the TPM public endorsement key.)	Use the public key in the TPM identity protocol.
Issuer unique identifier	Omit.	No action.
Subject unique identifier	Omit.	No action.
Extensions		
Authority key identifier	Assign "critical" the value FALSE. Assign the value of "subject key identifier" from the manufacturer's certificate, if available, else omit.	Use to locate the certificate that contains a public key of the manufacturer with which the signature on this certificate can be verified.
Subject key identifier	Omit.	No action.
Key usage	May be omitted. If included, then the key encipherment bit shall be set TRUE.	If present, then check that the key encipherment bit is TRUE, else reject.
Extended key usage	Omit.	If present and marked critical, then reject.
Private key usage period	Omit.	If present, then check that the current time is later than the notBefore time.
Certificate policies	Assign "critical" the value TRUE. Assign policyIdentifier at least one object identifier. Assign the cPSuri policy qualifier the value of an HTTP URL at which a plain language version of the TPM endorsement entity's certificate policy may be obtained. Assign the explicit text userNotice policy qualifier the value "TCPA Trusted Platform Module Endorsement".	Check that at least one acceptable policyIdentifier value is present. Transfer the acceptable policyInformation value to the TPM identity certificate "certificate policies" extension.
Policy mappings	Omit.	No action.
Subject alternative name	Assign "critical" the value FALSE. Include the TPM identity, using the directory name-form with RDNs for the TPM manufacturer, model and version numbers.	Check that the TPM manufacturer, model and version numbers are acceptable. Transfer to the TPM identify certificate "subject alternative name" extension value for the TPM.
Issuer alternative name	Omit.	No action.

Subject directory attributes	<p>Include a "subject directory attributes" extension. Assign "critical" the value FALSE. Include the multi-valued attribute "supported algorithms" (see X.509). Include object identifiers for the following algorithms: RSAES-OAEP, SHA-1 (1.3.14.3.2.26) and TPM identity protocol.</p> <p>Include the "TCPA Specification Version" attribute, with field values correctly reflecting the highest version of the TCPA specification with which the TPM implementation conforms.</p> <p>Optionally, include the "security qualities" attribute with a text string reflecting the security qualities of the TPM. (Note: this is the TPM distributed validation.)</p>	<p>Adapt the TPM identity protocol to use only algorithms supported by the TPM.</p> <p>Check that the TCPA specification version is acceptable, else reject.</p> <p>Optionally (and if present), check whether the TPM implementation has acceptable security qualities. Transfer to the TPM identity certificate "subject directory attributes" extension.</p>
Basic constraints	Assign "critical" the value TRUE. Assign "CA" the value FALSE	No action.
Name constraints	Omit.	No action.
Policy constraints	Omit.	No action.
Inhibit any policy	Omit.	No action.
CRL distribution points	Omit.	If present and marked critical, then reject.

## 9.5.2 Instantiation of Platform\_credentials

### **Start of informative comment:**

A platform certificate is an instantiation of a platform\_credential.

Access to the platform certificate must be restricted to entities that have a “need to know.” This is for reasons of privacy.

### **End of informative comment.**

If the data structure <platform\_certificate> is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.

### **Overview**

The Platform Endorsement Certificate represents an assertion by the platform endorsement entity that the referenced platform incorporates a TPM and an RTM in a manner that conforms with the TCPA specification.

### **Profile**

Note: some fields are assigned a value even though the certificate user performs no action with that value. In such cases, the intention is to inhibit non-TCPA implementations from making inappropriate use of the certificate.

In the case of the Platform endorsement certificate, the **issuer** is the platform manufacturer and the **user** is a Privacy CA.

Field	Issuer action	User action
Version	Assign value 1 (v2).	Check value = 1, else reject.
Holder	BaseCertificateID referencing the corresponding TPM endorsement certificate. (Note: this is the TPM credential reference.)	Check that the certificate ID correctly references the TPM endorsement certificate used to validate the TPM identity request message, else reject.
Issuer	The distinguished name of the platform endorsement entity. That is the entity that asserts that the subject platform incorporates a TPM and RTM in a manner that conforms with the TCPA specification. (Note: this may be the platform manufacturer or a conformance test laboratory.)	Check that the name is the name of one of the acceptable platform endorsement entities.
Signature	Assign algorithm identifier sha-1WithRSAEncryption (1:2:840:113549:1:1:5).	Check algorithm identifier = 1:2:840:113549:1:1:5, else reject. Validate the signature on the certificate using the public key of the Platform Endorsement Entity (which should be a 2048-bit RSA key), obtained by an out-of-band means and referenced by “issuer” and “authority key
Serial number	Assign a value unique amongst all certificates issued by “issuer”.	No action.
attrCertValidityPeriod	Assign notBefore to the current time and notAfter to a later time (maybe the latest time permitted by the	Check that the current time is later than the notBefore time, else reject.



Attributes	<p>encoding scheme).</p> <p>A "supported algorithms" attribute (see X.509) indicating the cryptographic algorithms supported by the platform.</p> <p>Include the "TCPA Specification Version" attribute, with field values correctly reflecting the highest version of the TCPA specification with which the platform implementation conforms.</p> <p>If the TPM has been successfully evaluated against a Common Criteria protection profile, then include the TPM protection profile identifier attribute.</p> <p>If the TPM has been successfully evaluated against a Common Criteria security target, then include the TPM security target identifier attribute.</p> <p>If the RTM and the means by which the TPM and RTM have been incorporated into the platform have been successfully evaluated against a Common Criteria protection profile, then include the "foundation protection profile" identifier attribute.</p> <p>If the RTM and the means by which the TPM and RTM have been incorporated into the platform have been successfully evaluated against a Common Criteria security target, then include the "foundation security target" identifier attribute.</p> <p>If there is, or will be, a Platform Conformance Certificate, then a ConformanceCertificateLocation attribute should be included to indicate how, and from where, it can be retrieved.</p> <p>Optionally, include the "security qualities" attribute with a text string reflecting the security qualities of the platform. (Note: this is the platform distributed validation.)</p>	<p>Transfer the object identifiers for any acceptable algorithms to the TPM identity certificate "subject directory attributes" extension.</p> <p>Check that the TCPA specification version is acceptable, else reject.</p> <p>Optionally, check whether the identifier is acceptable. Transfer the protection profile identifier to the TPM identity certificate.</p> <p>Optionally, check whether the identifier is acceptable. Transfer the security target identifier to the TPM identity certificate.</p> <p>Optionally, check whether the identifier is acceptable. Transfer the protection profile identifier to the TPM identity certificate "subject directory attributes" extension.</p> <p>Optionally, check whether the identifier is acceptable. Transfer the security target identifier to the TPM identity certificate "subject directory attributes" extension.</p> <p>Use the information to locate and retrieve the corresponding Platform Conformance Certificate.</p> <p>Optionally (and if present), check whether the platform implementation has acceptable security qualities. Transfer to the TPM identity certificate "subject directory attributes" extension.</p>
Issuer unique identifier	Omit.	No action.
Extensions		
Certificate policies	Assign "critical" the value TRUE. Assign policyIdentifier at least one	Check that at least one acceptable policyIdentifier value is present. Transfer the

policies		policyInformation value to the TPM identity certificate "certificate policies" extension.
Subject alternative name		Check that the manufacturer, model and version numbers are acceptable. Transfer to the TPM identity certificate "subject alternative name" extension.
Authority key identifier	from the platform endorsement entity certificate, if available, else omit.	The certificate user may use this value to locate the certificate that contains a public key of the platform endorsement entity with which the signature on this certificate can be verified.
SOA Identifier	Omit.	No action.
Authority Attribute Identifier	Omit.	No action.
Role Specification Certificate Identifier	Omit.	No action.
Basic Attribute Constraints	Assign "critical" the value TRUE. Assign "authority" the value FALSE.	Check that "authority" is FALSE.
Delegated Name Constraints	Omit.	No action.
Time Specification	Omit.	No action.
Acceptable Certificate Policies	Assign "critical" the value TRUE. Assign one or more of the values of policyIdentifier from the certificate policies extension of the TPM endorsement certificate.	Check that the certificate policies extension of the TPM endorsement certificate contains at least one of the values.
Attribute Descriptor	Omit.	No action.
User Notice	Omit.	No action.
No Rev Available	Omit.	No action.
Acceptable Privilege Policies	Omit.	No action.

### 9.5.3 Instantiation of TPM\_CONFORMANCE

#### Overview

The Platform Conformance Certificate represents an assertion that the referenced platform conforms with the TCPA specification.

#### Profile

Note: some fields are assigned a value even though the value is not used. In such cases, the intention is to inhibit non-TCPA use of the certificate.

In the case of the Platform conformance certificate, the *issuer* is a Privacy CA.

Field	Issuer action	
Version	Assign value 1 (v2).	
Holder	Include the platform name, uniquely identifying the type of the platform with RDNs for the manufacturer, model and version numbers.	
Issuer	The distinguished name of the platform conformance entity. That is the entity that asserts that the design of the platform conforms with the TCPA specification. (Note: this may be the platform manufacturer or a conformance test laboratory.)	
Signature	Assign algorithm identifier sha-1WithRSAEncryption (1:2:840:113549:1:1:5).	
Serial number	Assign a value unique amongst all certificates issued by "issuer".	No action.
attrCertValidityPeriod	Assign notBefore to the current time and notAfter to a later time (maybe the latest time permitted by the encoding scheme).	Check that the current time is later than the notBefore time, else reject.
Attributes	<p>Include a "supported algorithms" attribute (see X.509) indicating the algorithms supported by the platform.</p> <p>Include the "TCPA specification version" attribute, with field values correctly reflecting the highest version of the TCPA specification with which the platform implementation conforms.</p>	<p>Transfer the object identifiers for any acceptable algorithms to the TPM identity certificate "subject directory attributes" extension.</p> <p>Check that the TCPA specification version is acceptable, else reject.</p>

	<p>If the TPM has been successfully evaluated against a Common Criteria protection profile, then include the TPM protection profile identifier attribute.</p> <p>If the TPM has been successfully evaluated against a Common Criteria security target, then include the TPM security target identifier attribute.</p> <p>If the RTM and means by which the RTM and TPM are incorporated into the platform has been successfully evaluated against a Common Criteria protection profile, then include the foundation protection profile identifier attribute.</p> <p>If the RTM and the means by which the RTM and TPM have been incorporated into the platform have been successfully evaluated against a Common Criteria security target, then include the foundation security target identifier attribute.</p>	<p>Check that the identifier is acceptable. Transfer the protection profile identifier to the TPM identity certificate.</p> <p>Check that the identifier is acceptable. Transfer the security target identifier to the TPM identity certificate.</p> <p>Check that the identifier is acceptable. Transfer the protection profile identifier to the TPM identity certificate "subject directory attributes" extension.</p> <p>Check that the identifier is acceptable. Transfer the security target identifier to the TPM identity certificate "subject directory attributes" extension.</p>
Issuer unique identifier	Omit.	No action.
Extensions		
Certificate policies	Assign "critical" the value TRUE. Assign policyIdentifier at least one object identifier. Assign the cPSuri policy qualifier the value of an HTTP URL at which a plain language version of the platform conformance entity's certificate policy may be obtained. Assign the explicit text userNotice policy qualifier the value "TCPA Conformance Credential".	Check that at least one acceptable policyIdentifier value is present. Transfer the policyInformation value to the TPM identity certificate.
Subject alternative name	Assign "critical" the value FALSE. Include the platform name, uniquely identifying the type of the platform with RDNs for the platform manufacturer, model and version numbers.	Check that the manufacturer, model and version numbers are identical to those in the platform endorsement certificate "subject alternative name" extension.
Authority key identifier	Assign "critical" the value FALSE. Assign the value of "subject key identifier" from the platform conformance entity's public-key certificate, if available, else omit.	The certificate user may use this value to locate the certificate that contains a public key of the platform conformance entity with which the signature on this certificate can be verified.
SOA Identifier	Omit.	No action.
Authority Attribute	Omit.	No action.

Identifier		
Role Specification Certificate Identifier	Omit.	No action.
Basic Attribute Constraints	Assign "critical" the value TRUE. Assign "authority" the value FALSE.	Check that "authority" is FALSE.
Delegated Name Constraints	Omit.	No action.
Time Specification	Omit.	No action.
Acceptable Certificate Policies	Omit.	No action.
Attribute Descriptor	Omit.	No action.
User Notice	Omit.	No action.
No Rev Available	Omit.	No action.
Acceptable Privilege Policies	Omit.	No action.

### 9.5.4 Instantiation of Validation Certificate

**Start of informative comment:**

A "Validation Data Attribute Certificate" is an instantiation of validation data.

**End of informative comment.**

#### Overview

The validation data certificate represents an assertion by the component validation entity that the component instructions referenced by the certificate have the attributes conveyed in the certificate. The certificate syntax conforms with the X.509 definition for an attribute certificate.

In the case of the validation certificate, the **issuer** is the Validation Entity and the **user** is a TPS.

Field	Issuer action	User action
Version	Assign value 1 (v2).	Check value = 1, else reject.
Holder	ObjectDigestInfo with missing object identifier. The value of objectDigest shall be the digest calculated over the memory image of the software instructions using the identified digest algorithm.	Calculate the digest of the memory image of the software instructions and check that it is identical to the value in this field prior to passing control to the component, else reject.
Issuer	The distinguished name of the component validation entity. That is the entity that asserts that the component exhibits the attributes contained in the certificate. (Note: typically, but not necessarily, the manufacturer of the component).	Check that the name is the name of one of the acceptable component validation entities.
Signature	Assign algorithm identifier sha-1WithRSAEncryption (1:2:840:113549:1:1:5).	Check algorithm identifier = 1:2:840:113549:1:1:5, else reject. Validate the signature on the certificate using the public key of the software manufacturer (which should be a 2048-bit RSA key), obtained by an out-of-band means and referenced by "issuer" and "authority key identifier".
Serial number	Assign a value unique amongst all certificates issued by "issuer".	No action.
attrCertValidityPeriod	Assign notBefore to the current time and notAfter to a later time (maybe the latest time permitted by the encoding scheme).	Check that the current time is later than the notBefore time, else reject.
Attributes	Include the "TCPA specification version" attribute, with field values correctly reflecting the highest version of the TCPA specification with which the component conforms.	Check that the TCPA specification version is acceptable, else reject.
	Optionally, include the "security qualities" attribute with a text string reflecting the security qualities of the component. (Note: this is the component distributed validation.)	Optionally (and if present), check whether the component implementation has acceptable security qualities.

Issuer unique identifier	Omit.	No action.
Extensions		
Certificate policies	Assign "critical" the value TRUE. Assign policyIdentifier at least one object identifier. Assign the cPSuri policy qualifier the value of an HTTP URL at which a plain language version of the component conformance entity's certificate policy may be obtained. Assign the explicit text userNotice policy qualifier the value "TCPA Validation	Check that at least one acceptable policyIdentifier value is present.
Subject Alternative Name	Assign "critical" the value FALSE. Include the component name, using the "component name" attribute, with RDNs for the component manufacturer, model and version numbers.	May be used to determine whether or not the component is trustworthy.
Authority key identifier	Assign "critical" the value FALSE. Assign the value of "subject key identifier" from the component validation entity certificate, if available, else omit.	The certificate user may use this value to locate the certificate that contains a public key of the component validation entity with which the signature on this certificate can be verified.
SOA Identifier	Omit.	No action.
Authority Attribute Identifier	Omit.	No action.
Role Specification Certificate Identifier	Omit.	No action.
Basic Attribute Constraints	Assign "critical" the value TRUE. Assign "authority" the value FALSE.	Check that "authority" is FALSE.
Delegated Name Constraints	Omit.	No action.
Time Specification	Omit.	No action.
Acceptable Certificate Policies	Omit.	No action.
Attribute Descriptor	Omit.	No action.
User Notice	Omit.	No action.
No Rev Available	Omit.	No action.
Acceptable Privilege Policies	Omit.	No action.





### 9.5.5 Instantiation of TPM\_IDENTITY\_CREDENTIAL

**Start of informative comment:**

A TPM identity certificate is an instantiation of a TPM\_IDENTITY\_CREDENTIAL.

Access to the TPM identity certificate must be restricted to entities that have a “need to know.” This is for reasons of privacy.

This definition assumes that TPM identity keys are 2048bit RSA keys.

**End of informative comment.**

If the data structure <TPM identity certificate> is stored on a platform after an Owner has taken ownership of that platform, it SHALL exist only in storage to which access is controlled and is available to authorized entities.

#### Overview

The TPM identity certificate represents an assertion by the Privacy CA that the referenced TPM identity is controlled by a TPM that conforms with the TPM specification. It contains a different public key to that contained in the TPM endorsement certificate, but it contains identifying and policy information transferred from the TPM endorsement, platform endorsement and platform conformance certificates.

#### Profile

Note:

- Some fields are assigned a value even though the certificate user performs no action with that value. In such cases, the intention is to inhibit non-TCPA implementations from making inappropriate use of the certificate.
- The policies identified in the TPM and platform certificates are represented by oids and are not distinguishable except by reference to the contents of the policies themselves. The verifier, however, must be able to distinguish between the different policy types.

In the case of the TPM identity certificate, the **issuer** is the Privacy CA and the **user** is an integrity verifier.

Field	Issuer action	User action
Version	Assign value 2 (v3).	Check value = 2, else reject.
Serial number	Assign a value unique amongst all certificates issued by “issuer”.	No action.
Signature	Assign algorithm identifier sha-1WithRSAEncryption (1:2:840:113549:1:1:5).	Check the algorithm identifier = 1:2:840:113549:1:1:5, else reject. Validate the signature on the certificate using the public key of the Privacy CA (which should be a 2048-bit RSA key), obtained by an out-of-band means and referenced by “issuer” and “authority key identifier”.
Issuer	The distinguished name of the Privacy CA.	Check that the name is the name of an acceptable Privacy CA.
Validity	Assign notBefore to the current time and notAfter to a later time (maybe the latest time permitted by the encoding scheme).	Check that the current time is later than the notBefore time, else reject.
Subject	NULL.	No action.
Subject public	Assign algorithm identifier sha-	Check algorithm identifier =

key info	1WithRSAEncryption (1:2:840:113549:1:1:5). The 2048-bit RSA public key provided to the Privacy CA by the TPM owner in the identity request message.	1:2:840:113549:1:1:5, else reject. Use the public key in the integrity verification procedure.
Issuer unique identifier	Omit.	No action.
Subject unique identifier	Omit.	No action.
Extensions		
Authority key identifier	Assign "critical" the value FALSE. Assign the value of "subject key identifier" from the Privacy CA's public-key certificate, if available, else omit.	The certificate user may use this value to locate the certificate that contains a public key of the Privacy CA with which the signature on this certificate can be verified.
Subject key identifier	Omit.	No action.
Key usage	May be omitted. If included, then the digital signature bit shall be set TRUE.	If present, then check that the digital signature bit is TRUE, else reject.
Extended key usage	Omit.	If present and marked critical, then reject.
Private key usage period	Omit.	If present, then check that the current time is later than the notBefore time, else reject.
Certificate policies	Assign "critical" the value TRUE. Assign policyIdentifier at least one object identifier. Optionally, assign the cPSuri the value of an HTTP URL at which a plain language version of the Privacy CA's certificate policy may be obtained. Assign the explicit text userNotice policy qualifier the value "TCPA Trusted Platform Identity". Also, include the policyInformation values from the certificate policies extensions of the TPM endorsement and platform endorsement and conformance certificates provided in the TPM identity request message.	Check that at least one acceptable Privacy CA policyIdentifier value is present. Optionally, check that at least one acceptable TPM endorsement, one acceptable platform endorsement and one acceptable platform conformance policyIdentifier value are present.
Policy mappings	Omit.	No action.
Subject alternative name	Assign "critical" the value FALSE. Include three values in the extension:  The TPM manufacturer, model and version numbers from the TPM endorsement certificate "subject alternative name" extension provided in the TPM identity request message;  The platform manufacturer, model	Check that the manufacturer, model and version numbers of the TPM and of the platform are acceptable.

<p>Issuer alternative name</p> <p>Subject directory attributes</p>	<p>and version numbers from the platform endorsement certificate "subject alternative name" extension provided in the TPM identity request message; and</p> <p>The TPM identity label provided to the Privacy CA by the TPM owner in the identity request message, encoded as a TPMIdLabel other-name. The TPM owner should choose a label syntax and semantics that are understood by the integrity verifier. (Note: the specified syntax accommodates multi-byte character sets).</p> <p>Omit.</p> <p>Assign "critical" the value FALSE. Include a multi-valued "supported algorithms" (see X.509) attribute containing object identifiers from the "subject directory attributes" extension of the TPM endorsement certificate and the "attributes" field of the platform endorsement certificate and the platform conformance certificate provided in the TPM identity request message.</p> <p>Include the single-valued "TPM protection profile" attribute from the platform endorsement certificate provided in the TPM identity request message.</p> <p>Include the single-valued "TPM security target" attribute from the platform endorsement certificate provided in the TPM identity request message.</p> <p>Include the single-valued "Foundation protection profile" attribute from the platform endorsement certificate provided in the TPM identity request message.</p> <p>Include the single-valued "Foundation security target" attribute from the platform endorsement certificate provided in the TPM identity request message.</p> <p>Include the "security qualities" attribute from the TPM endorsement certificate provided in the TPM identity request message. (Note: this is the</p>	<p>No action.</p> <p>Adapt the integrity verification protocol to use only algorithms supported by the TPM and the associated platform.</p> <p>Check that the identifier is acceptable.</p> <p>Check that the identifier is acceptable.</p> <p>Check that the identifier is acceptable.</p> <p>Check that the identifier is acceptable.</p> <p>Optionally (and if present), check whether the TPM has acceptable security qualities.</p>
--	---	--

	TPM distributed validation.)	
	Include the "security qualities" attribute from the platform endorsement certificate provided in the TPM identity request message. (Note: this is the platform distributed validation.)	Optionally (and if present), check whether the platform has acceptable security qualities.
	Include the "tcpaVersion" attribute provided in the TPM identity request message.	Check that the TCPA specification version is acceptable, else reject.
Basic constraints	Assign "critical" the value TRUE. Assign "CA" the value FALSE.	No action.
Name constraints	Omit.	No action.
Policy constraints	Omit.	No action.
Inhibit any policy	Omit.	No action.
CRL distribution points	Omit.	If present and marked critical, then reject.

### 9.5.6 ASN.1 Definitions

***Start of informative comment:***

The intention is to register TCPA as an "international body" in the ISO registration hierarchy. This will lead to shorter oids (object identifiers) and gives TCPA autonomy in the management of its own object identifiers.

***End of informative comment.***

The syntax of the "security qualities" attribute is as follows:

```
SecurityQualities ATTRIBUTE ::= {
    WITH SYNTAX SecurityQualities
    ID tcpa-tpmSecurityQualities }

SecurityQualities ::= SEQUENCE {
    version INTEGER, --0 for this version of the attribute syntax --
    statement [0]    UTF8String }
```

Note: future versions of this certificate profile may define additional, optional, "security qualities" fields. Inclusion of the "statement" field will remain mandatory.

The syntax of the "TCPA Specification Version" attribute is as follows:

```
TCPASpecVersion ATTRIBUTE ::= {
    WITH SYNTAX TCPASpecVersion
    ID tcpa-specVersion }

TCPASpecVersion ::= SEQUENCE {
    major INTEGER,
    minor INTEGER }
```

The syntax of the protection profile and security target attributes is as follows:

```
TPMProtectionProfile ATTRIBUTE ::= {
    WITH SYNTAX ProtectionProfile
    ID tcpa-at-tpmProtectionProfile }

TPMSecurityTarget ATTRIBUTE ::= {
    WITH SYNTAX SecurityTarget
    ID tcpa-at-tpmSecurityTarget }

FoundationProtectionProfile ATTRIBUTE ::= {
    WITH SYNTAX ProtectionProfile
    ID tcpa-at-foundationProtectionProfile }

FoundationSecurityTarget ATTRIBUTE ::= {
    WITH SYNTAX SecurityTarget
    ID tcpa-at-foundationSecurityTarget }
ProtectionProfile ::= OBJECT IDENTIFIER
SecurityTarget ::= OBJECT IDENTIFIER
```

The syntax of the "component name" attribute is as follows:

```
ComponentName ATTRIBUTE ::= {
    WITH SYNTAX Name
    ID tcpa-at-componentName }
```

The following definitions define the syntax of the RDNs used in the subject alternative name extension to identify the type of the TPM and the platform.

```
TpmManufacturer ATTRIBUTE ::= {
    WITH SYNTAX UTF8String
    ID tcpa-at-tpmManufacturer }
```

```
TpmModel ATTRIBUTE ::= {
    WITH SYNTAX UTF8String
    ID tcpa-at-tpmModel }
```

```
TpmVersion ATTRIBUTE ::= {
    WITH SYNTAX UTF8String
    ID tcpa-at-tpmVersion }
```

```
PlatformManufacturer1 ATTRIBUTE ::= {
    WITH SYNTAX UTF8String
    ID tcpa-at-platformManufacturer }
```

```
PlatformModel ATTRIBUTE ::= {
    WITH SYNTAX UTF8String
    ID tcpa-at-platformModel }
```

```
PlatformVersion ATTRIBUTE ::= {
    WITH SYNTAX UTF8String
    ID tcpa-at-platformVersion }
```

```
TPMIdLabel OTHER-NAME ::= {UTF8String IDENTIFIED BY {tcpa-at-tpmIdLabel}}
```

--Object identifier assignments--

tcpa	OBJECT IDENTIFIER ::= {TBD}
tcpa-specVersion	OBJECT IDENTIFIER ::= {tcpa-1}
tcpa-attribute	OBJECT IDENTIFIER ::= {tcpa-2}
tcpa-protocol	OBJECT IDENTIFIER ::= {tcpa-3}
tcpa-at-tpmManufacturer	OBJECT IDENTIFIER ::= {tcpa-attribute 1}
tcpa-at-tpmModel	OBJECT IDENTIFIER ::= {tcpa-attribute 2}
tcpa-at-tpmVersion	OBJECT IDENTIFIER ::= {tcpa-attribute 3}
tcpa-at-platformManufacturer	OBJECT IDENTIFIER ::= {tcpa-attribute 4}
tcpa-at-platformModel	OBJECT IDENTIFIER ::= {tcpa-attribute 5}
tcpa-at-platformVersion	OBJECT IDENTIFIER ::= {tcpa-attribute 6}
tcpa-at-componentManufacturer	OBJECT IDENTIFIER ::= {tcpa-attribute 7}
tcpa-at-componentModel	OBJECT IDENTIFIER ::= {tcpa-attribute 8}
tcpa-at-componentVersion	OBJECT IDENTIFIER ::= {tcpa-attribute 9}
tcpa-at-securityQualities	OBJECT IDENTIFIER ::= {tcpa-attribute 10}
tcpa-at-tpmProtectionProfile	OBJECT IDENTIFIER ::= {tcpa-attribute 11}
tcpa-at-tpmSecurityTarget	OBJECT IDENTIFIER ::= {tcpa-attribute 12}
tcpa-at-foundationProtectionProfile	OBJECT IDENTIFIER ::= {tcpa-attribute 13}
tcpa-at-foundationSecurityTarget	OBJECT IDENTIFIER ::= {tcpa-attribute 14}
tcpa-at-tpmIdLabel	OBJECT IDENTIFIER ::= {tcpa-attribute 15}
tcpa-prt-tpmIdProtocol	OBJECT IDENTIFIER ::= {tcpa-protocol 1}

## 10. Conformance Criteria

### 10.1 Base Levels for Interoperability

***Start of informative comment:***

The TCPA Support Services (TSS) will interoperate with other TSS devices and applications external to the TPM. The functions that interoperate are identity creation, challenge and response; backup; and maintenance. The interoperability must be at a level so that an application or other TSS can, without modification, send messages and receive replies. The messaging system may be either real-time or store-and-forward.

The use of TPM and TSS is intentional in the conformance section. The difference between the two is the level of protection that is available for the functions or data. The TPM provides tight control over execution and data access, but for the TSS there is no such requirement.

To achieve maximum flexibility the TSS supports a negotiation protocol. This protocol allows the requestor to determine which features are available and the parameter settings that are appropriate for each of them.

There is no guarantee of interoperability when support for additional algorithms and protocols is provided.

***End of informative comment.***

The algorithms and protocols in this specification are the REQUIRED algorithms and protocols. A TPM subsystem MAY support additional algorithms and protocols. When this specification specifies the use of the TSS for a feature, an implementation MAY place the feature in the TPM.

### 10.2 Conformance Specification Sheet

***Start of informative comment:***

This section provides a quick listing of the protocols and algorithms that a TPM must support. For details review the section specific to the function in question.

**Algorithms**

- RSA, SHA-1, HMAC

**Operations**

- Random number generation
- Key generation
- Digital signatures (signing and verification)
- Protected storage
- Auditing
- Volatile memory
- Non-volatile memory

***End of informative comment.***

### 10.3 Protocol Negotiation and Algorithm Agility

***Start of informative comment:***

The TPM requires interoperability between devices when sending migration packets, identities and backup issues. For these reasons the specification mandates algorithms and message formats.

A related issue is that the set of algorithms picked by the specification may not meet the needs of a certain community. The specification therefore allows different algorithms to be in use. For instance, when creating an identity the creator can specify the algorithm and algorithm parameters for the identity. The specification requires that the TPM support the RSA algorithm, however the TPM may support additional algorithms and parameters.

Any challenger can request the list of algorithms and parameters that a TPM supports using the TPM\_GetCapability command.

A challenger does not negotiate algorithms and parameters rather the challenger requests a specific type and the TPM either executes the command or fails the request.

***End of informative comment.***

The TPM MUST support the base algorithms specified for each operation. The TPM MAY support additional algorithms and parameters.

The TPM manufacturer MUST include in the TPM credential all algorithms that the TPM supports.

The TSS manufacturer MUST include in the platform credential all algorithms that the TSS supports.

## 10.4 Cryptographic Algorithms and Protocols

***Start of informative comment:***

The algorithms and protocols are the minimum that the TSS and TPM must support. Additional algorithms and protocols may be available to the TPM and TSS. All algorithms and protocols available in the TPM and TSS must be included in the list in the TPM and platform credential.

***End of informative comment.***

### 10.4.1 Asymmetric

***Start of informative comment:***

The asymmetric algorithm provides both digital signatures and wrapping of keys. The requirement of the TPM to support RSA allows the specification of one algorithm for both purposes.

TPM devices that implement different algorithms may have different algorithms perform the signing and wrapping.

There is no requirement concerning how the RSA algorithm is to be implemented. TPM manufacturers may use Chinese Remainder Theorem (CRT) implementations or any other method. Designers should review P1363 for guidance on RSA implementations.

***End of informative comment.***

- The TPM MUST support RSA.
- The TPM MUST use the RSA algorithm for encryption and digital signatures.
- The TPM MUST support key sizes of 512, 1024, and 2048 bits. The TPM MAY support other key sizes. The minimum RECOMMENDED key size is 1024 bits.
- The RSA public exponent MUST be  $e$ , where  $e = 2^{16} + 1$ .

TPM devices that use CRT as the RSA implementation MUST provide protection and detection of failures during the CRT process to avoid attacks on the private key.

The TPM MAY implement other asymmetric algorithms such as DSA or elliptic curve. These algorithms may be in use for wrapping, signatures, and other operations. There is no guarantee that these keys can migrate to other TPM devices or that other TPM devices will accept signatures from these additional algorithms.



### 10.4.2 Symmetric

***Start of informative comment:***

The encryption done by the TPM does not require a symmetric algorithm. The TSS must provide the bulk encryption support. The assumption is that the TSS has larger bandwidth and more MIPS to accomplish this type of encryption.

There is no requirement that a TPM NOT support a symmetric algorithm. A TPM may implement a symmetric algorithm.

The requirement to support both DES and 3DES is because some localities have restrictions on the import or export of 3DES and the TSS should not have an export or import limitation. DES should be in use only when the 3DES is not allowable.

***End of informative comment.***

The TSS MUST support 3DES. 3DES SHOULD be the symmetric algorithm of choice. The key size of 3DES MUST be 196 bits (three 64-bit keys). 3DES MUST be run in encrypt-decrypt-encrypt (EDE) mode. The TSS MUST provide detection of weak 3DES keys.

The TSS MUST support DES. The key size for DES MUST be 64 bits (56 bits plus parity). The TSS MUST provide detection of weak DES keys.

The TSS SHOULD have support for AES when it becomes available.

A TPM MUST support the storage of at least 256-bit symmetric keys.

### 10.4.3 Hashing

The TPM MUST support the SHA-1 hash algorithm as defined by FIPS-181. The output of SHA-1 is 160 bits and all areas that expect a hash value are REQUIRED to support the full 160 bits.

### 10.4.4 Signature Operations

The TPM MUST use the RSA algorithm for signature operations.

The TPM MAY use other asymmetric algorithms for signatures; however, there is no requirement that any other TPM device either accept or verify those signatures.

The TPM MUST use P1363 for the format and design of the signature output.

### 10.4.5 Creating a PCR composite hash

The definition specifies the operation necessary to create TCPA\_COMPOSITE\_HASH.

#### Action

The hashing MUST be done using the SHA-1 algorithm.

The input must be a valid TCPA\_PCR\_SELECTION structure.

The process creates a TCPA\_PCR\_COMPOSITE structure from the TCPA\_PCR\_SELECTION structure and the PCR values to be hashed. If constructed by the TPM the values MUST come from the current PCR registers indicated by the PCR indices in the TCPA\_PCR\_SELECTION structure.

The process then computes a SHA-1 digest of the TCPA\_PCR\_COMPOSITE structure.

The output is the SHA-1 digest just computed.

### 10.4.6 Using Secret Keys

#### **Informative comments:**

Secret keys can be loaded into a TPM, but preferably are generated inside the TPM.

A TPM generated key must not be used as a secret key if it has already been exposed.

Secret keys obtained from blobs must not be exposed outside the TPM.

#### **End of informative comments.**

A secret key is a key that is a private asymmetric key or a symmetric key.

Data SHOULD NOT be used as a secret key by a TCPA protected capability unless that data has been extant only in a shielded location.

A key generated by a TCPA protected capability SHALL NOT be used as a secret key unless that key has been extant only in a shielded location.

A secret key obtained by a TCPA protected capability from a Protected Storage blob SHALL be extant only in a shielded location.

## 10.5 Random Number Generator (RNG)

### ***Start of informative comment:***

The Random Number Generator (RNG) is the source of randomness in the TPM. The TPM uses these random values for nonces, key generation and randomness in signatures.

The understanding is that this definition of the RNG, depending on implementation, could be a Pseudo Random Number Generator (PRNG). On those devices that have a hardware source of entropy, this implementation may be an RNG and not a PRNG so there is no need for to keep track of which is which; that is, the specification will always use RNG.

### ***End of informative comment.***

The RNG for the TPM will consist of the following components:

- Entropy source and collector
- State register
- Mixing function

The RNG capability is a TPM-protected capability with no access control.

The RNG output may or may not be shielded data. When the data is for internal use by the TPM (e.g., asymmetric key generation) the data **MUST** be held in a shielded location. When the data is for use by the TSS or another external caller, the data is not shielded.

### 10.5.1 Entropy Source and Collector

#### ***Start of informative comment:***

The entropy source is the process or processes that provide entropy. These types of sources could include noise, clock variations, air movement, and other types of events.

The entropy collector is the process that collects the entropy, removes bias, and smoothes the output. The difference between the collector and the mixing function (described in section 10.6.3, "Mixing Function") is that the collector may have special code to handle any bias or skewing of the raw entropy data. For instance, if the entropy source has a bias of creating 60 percent 1s and only 40 percent 0s, then the collector design takes that bias into account before sending the information to the state register.

#### ***End of informative comment.***

The entropy source **MUST** provide entropy to the state register in a manner that provides entropy that is not visible to an outside process. For compliance purposes, the entropy source **MAY** be in the TSS and not the TPM; however, attention **MUST** be paid to the reporting mechanism.

The entropy source **MUST** provide the information only to the state register. The entropy source may provide information that has a bias, so the entropy collector must remove the bias before updating the state register. The bias removal could use the mixing function or a function specifically designed to handle the bias of the entropy source. The entropy source can be a single device (such as hardware noise) or a combination of events (such as disk timings). It is the responsibility of the entropy collector to update the state register whenever the collector has additional entropy.

### 10.5.2 State Register

#### ***Start of informative comment:***

The state register implementation may use two registers: a non-volatile register and a volatile register. The TPM loads the volatile register from the non-volatile register on startup. Each subsequent change to the state register from either the entropy source or the mixing function affects the volatile state register. The TPM saves the current value of the volatile state register to the non-volatile register on TPM power-

down. The TPM may update the non-volatile register at any other time. The reasons for using two registers are

- to handle an implementation in which the non-volatile register is in a flash device and
- to avoid overuse of the flash, as the number of writes to a flash device are limited.

***End of informative comment.***

The state register is in a TPM-shielded location. The state register **MUST** be non-volatile. The update function to the state register is a TPM-protected capability. The primary input to the update function **SHOULD** be the entropy collector.

If the current value of the state register is unknown, calls made to the update function with known data **MUST NOT** result in the state register ending up in a state that an attacker could know. This requirement implies that the addition of known data **MUST NOT** result in a decrease in the entropy of the state register.

The TPM **MUST NOT** export the state register.

### 10.5.3 Mixing Function

***Start of informative comment:***

The mixing function takes the state register and produces some output.

The mixing function is a TPM-protected capability. The mixing function takes the state register and creates the output of the RNG. The output **MUST** conform to the requirements for PRNG from FIPS 140-1.

***End of informative comment.***

Each use of the mixing function **MUST** affect the state register. This requirement is to affect the volatile register and does not need to affect the non-volatile state register.

### 10.5.4 RNG Reset

***Start of informative comment:***

The resetting of the RNG occurs at least in response to a loss of power to the device.

These tests prove only that the RNG is still operating properly; they do not prove how much entropy is in the state register. This is why the self-test checks only after the load of previous state and may occur before the addition of more entropy.

***End of informative comment.***

The RNG **MUST NOT** output any bits after a system reset until the following occurs:

- The entropy collector performs an update on the state register. This does not include the adding of the previous state but requires at least one bit of entropy.
- The mixing function performs a self-test. This self-test **MUST** occur after the loading of the previous state. It **MAY** occur before the entropy collector performs the first update.

## 10.6 Key Generation

***Start of informative comment:***

Key generation is algorithm-specific. The requirements for a given algorithm come from the preceding section or sections specific to it.

There are no timing requirements on the length of time that a TPM must meet when performing key generation.

***End of informative comment.***

### 10.6.1 Asymmetric

The TPM MUST generate asymmetric key pairs. The generate function is a protected capability and the private key is held in a shielded location. The implementation of the generate function MUST be in accordance with P1363.

The prime-number testing for the RSA algorithm MUST use the definitions of P1363. If additional asymmetric algorithms are available, they MUST use the definitions from P1363 for the underlying basis of the asymmetric key (for example, elliptic curve fitting).

### 10.6.2 Symmetric

The TSS MUST generate a symmetric key by taking the next  $n$  bits from the TPM RNG.

The TSS SHOULD provide any processing of a symmetric key. Processing is an algorithm-specific operation and implementation is left to the designer.

### 10.6.3 Nonce Creation

The creation of all nonce values MUST use the next  $n$  bits from the TPM RNG.

## 10.7 Auditing

#### ***Start of informative comment:***

The TPM and TSS must be able to report a log of events. The log uses the same paradigm as the PCRs, the TPM keeps a PCR value that extends for each log event, and the TSS maintains the log entries for Challengers to review.

The TPM generates an audit event and the TSS creates the log. The protection of the log is a TSS requirement. The TSS is responsible for collecting each audit log event.

The TPM uses a PCR and extends it for each audit event. The TSS can use the PCR to create a log that shows any attempt to tamper with it.

The TPM Owner can select the operations that will generate an audit event.

#### ***End of informative comment.***

The TPM MUST be able to generate audit events for all TCPA protected capabilities.

The TPM Owner MUST be able to select the functions that will generate an audit event at any time.

The TPM MUST provide a PCR to store and log the audit events. The TPM MUST allow for the reporting of the current audit log PCR value. The value that the TPM adds to the TPM audit PCR MUST be the TCPA\_AUDIT\_EVENT structure.

The TSS MUST provide a log of all TPM-generated events. The TPM will generate the event and the TSS will fill in the event details. The TPM SHALL provide as much detail as it has available; however, the TSS MUST fill in all remaining details for the audit event. For instance, the audit event will require a data and time stamp on the event. There is no requirement for a clock function in the TPM, so the date and time would come normally from the TSS.

The TPM MAY generate audit events for other functions and activities not on this list.

## 10.8 Self-Tests

The TPM MUST provide startup self-tests. The TPM MUST provide mechanisms to allow the self-tests to be run on demand. The response from the self-tests is pass or fail.

The TPM MUST complete the startup self-tests in a manner and timeliness that allows the TPM to be of use to the BIOS during the collection of integrity metrics. The TPM MUST complete the required checks

before a given feature is in use. This requirement allows the TPM to test the integrity metric storage and allow its use while simultaneously continuing to test the signature engine.

There are two sections of startup self-tests: required and recommended. The recommended tests are not a requirement due to timing constraints. The TPM manufacturer should perform as many tests as possible in the time constraints.

The TPM **MUST** report the tests that it performs.

The TPM **MUST** provide a mechanism to allow for any self-test to execute on request by any Challenger. The testing can be the entire suite of tests or an individual test.

The TPM **MUST** provide for testing of some operations during each execution of the operation.

### **10.8.1 Required Self-Tests**

The TPM **MUST** check the following:

- RNG functionality. This test follows FIPS 140-1, which checks the functioning of an RNG.
- Reading and extending the integrity registers. The self-test for the integrity registers will leave the integrity registers in a known state.
- Endorsement key pair integrity. This requirement specifies that the TPM will verify that the endorsement key pair can sign and verify a known value. This test also tests the RSA sign and verify engine.
- The integrity of the protected capabilities of the TPM. This means that the TPM must ensure that its “microcode” has not changed, and not that a test must be run on each function.
- Any tamper-resistance markers. The tests on the tamper-resistance or tamper-evident markers are under programmable control. There is no requirement to check tamper-evident tape or the status of epoxy surrounding the case.

### **10.8.2 Recommended Checks**

The TPM **SHOULD** check the following:

- The hash functionality. This check will hash a known value and compare it to an expected result. There is no requirement to accept external data to perform the check. The TPM **MAY** support a test using external data.
- Any symmetric algorithms. This check will use known data with a random key to encrypt and decrypt the data.
- Any additional asymmetric algorithms. This check will use known data to encrypt and decrypt.
- The key-wrapping mechanism. The TPM should wrap and unwrap a key. The TPM **MUST NOT** use the endorsement key pair for this test.

### **10.8.3 Self-Test Failure**

When the TPM detects a failure during any self-test, the part experiencing the failure **MUST** enter a shut-down mode. This shut-down mode will allow only the following operation to occur:

- Update. The update function must be available to recover the addition of invalid microcode.

All other operations will return the error code `TCPA_FAILEDSELFTEST`.

## **10.9 Object Reuse**

The TPM **MUST** destroy and erase all temporal objects when the TPM finishes processing the object. The use of an object can be a long-term operation. For instance, the TPM could load an identity key and keep the key in memory while performing multiple challenge and response operations. There is no requirement

to unload the object after each operation, but there is a requirement that the object be properly disposed of when all operations are complete.

When an internal TPM process uses objects, no information regarding the object may be available to outside processes. The TPM **MUST** enforce access control to all objects carrying sensitive information.

## 10.10 Maintenance

### *Start of informative comment:*

The maintenance feature is a vendor-specific feature, and its implementation is vendor-specific. The implementation must, however, meet the minimum requirements as defined in section 7.2.14 so that one implementation of the maintenance feature does not provide a hole into the TCPA system.

There is no requirement that the maintenance feature be available, but if it is implemented, then the requirements must be met.

The maintenance feature described in the specification is an example only, and not the only mechanism that a manufacturer could implement that meets these requirements.

### *End of informative comment.*

The maintenance feature **MUST** ensure that the information can be on only one TPM at a time. Maintenance **MUST** ensure that at no time the process will expose a shielded location. Maintenance **MUST** require the active participation of the Owner.

## 10.11 Backup

### *Start of informative comment:*

The purpose of backup is to take a key and move it to another TPM. The backup mechanism must move only migratable information.

The blob that the backup feature creates must be usable by any other TPM. This requirement holds only for keys and data that are usable by all TPMs. For example, there is no requirement that a 768-bit RSA key be acceptable by all TPM devices. The migration of information has a guarantee only when the key uses one of the required sizes.

### *End of informative comment.*

The TPM **MUST** support the backup feature. The TPM **MUST** create a blob of migratable data that is readable by any other TPM. A receiving TPM **MAY** reject a backup blob if the underlying information is a non-standard size or algorithm.

## 10.12 Strength of Function

### *Start of informative comment:*

The common criteria defines Strength of Function (SOF) as a qualification of a Target of Evaluation (TOE) security function expressing the minimum efforts assumed necessary to defeat its expected security behavior by directly attacking its underlying security mechanisms.

Here are some definitions for the common SOF criteria:

- **SOF-basic.** A level of the TOE SOF where analysis shows that the function provides adequate protection against casual breach of TOE security by attackers possessing a low attack potential.
- **SOF-medium.** A level of the TOE SOF where analysis shows that the function provides adequate protection against straightforward or intentional breach of TOE security by attackers possessing a moderate attack potential

- **SOF-high.** A level of the TOE SOF where analysis shows that the function provides adequate protection against a deliberately planned or organized breach of TOE security by attackers possessing a high attack potential

There is no single overall SOF definition; instead, each operation needs a review of what the SOF should be. The Protection Profile will specify the SOF for each operation, command, function, and so on.. For instance, the SOF for protection of the endorsement key pair will be SOF-high, but the SOF for tamper resistance will be SOF-basic.

The testing lab will determine if a specific security target implementation of the Protection Profile meets the SOF level. This specification will not specify definition of the SOF as this metric is an ever-changing value. That is, what was high a few years ago is now not even at the basic level. It is certainly possible that a device that receives certification will not pass given changes in the SOF definition in the future.

***End of informative comment.***

The TPM MUST report the SOF values to a Challenger and the SOF values MUST be part of the TPM endorsement certificate and the platform conformance certificate.

## 10.13 Protection Profile

***Start of informative comment:***

The TCPA specification will use two Protection Profiles to judge conformance with the specification. They are the TCPA Trusted Platform Module Protection Profile (TCPA-TPMPP) and the TCPA Trusted Platform Connection Protection Profile (TCPA-TPCPP).

The TPMPP provides the evaluation of a TPM. The security targets that reference this Protection Profile will provide the mechanism for platform manufacturers to judge between different TPM providers. The TOE for the TPMPP covers just the TPM and does not include any TSS functionality.

The TPCPP provides the evaluation of the connection of the TPM to the platform and the connection of the RMT to the platform and TPM. The security targets that reference this Protection Profile will provide the mechanism for platform purchasers the ability to judge between different platforms. The TOE for the TPCPP will include the TPMPP.

The Protection Profiles are separate documents and refer back to this specification. The following discussion of the Protection Profiles is for reference only, and the actual text of the profiles supersedes any comments in this section.

The basis of the Protection Profiles is the attack tree that shows the threats against the TPM and TSS. The attack tree is a separate document that is an inherent part of this specification. The basic design point for the attack tree is that the TPM should be resistant to all software attacks and somewhat resistant to hardware attacks.

***End of informative comment.***

## 10.14 Compliance to Specification

***Start of informative comment:***

The TCPA does not evaluate compliance to this specification directly. The evaluation of compliance to the specification comes from the manufacturer creating a security target that meets the Protection Profile (either TPMPP or TPSP).

After the TCPA creates a Protection Profile, each manufacturer has the option of creating a security target to evaluate against the Protection Profile. This security target is implementation-specific and could cover either a machine or an application using the profile.

The evaluation of a security target provides assurances to the buying public that the manufacturer has created a secure interoperable system.

***End of informative comment.***



## 10.15 Field Upgrade

***Start of informative comment:***

A TPM, once in the field, may have need to update the protected capabilities. This command, which is optional, provides the mechanism to perform the update.

***End of informative comment.***

The TPM SHOULD have provisions for upgrading the subsystem after shipment from the manufacturer. If provided the mechanism MUST follow the requirement from section 8.15 .

## 10.16 Physical Presence or Access

***Start of informative comment:***

This specification includes commands which require "local" or "physical" presence at the platform before the command will operate. The intention is that these commands cannot be activated without authorization provided by direct interaction with a person

It must be possible to control a TPM. Such controls include those to clear an existing Owner from the TPM, temporarily deactivate a TPM, and temporarily disable a TPM. Some such commands must work without conventional authorization information, because they will be required when the necessary authorization information is unavailable (because there is no Owner or because the authorization information has been lost). Such commands are subject to "denial of service" attacks, and ideally require other forms of authorization

Some commands are therefore prescribed to require physical presence (of a person) at the platform before the command will operate. Such commands could be authorised with or by purely physical or electrical methods, or with or by physical presence detected using software when the platform is in a restricted state. Such authorization is difficult or impossible to reproduce by rogue software, depending on the exact method of implementation. The actual method of implementation of such authorization is the choice of the manufacturer. The overall strength of such authorization is reflected in the "security target" of the platform.

In a PC, such authorization might be implemented using direct electrical connections from a switch, or using software during the POST

***End of informative comment.***

The requirement for physical presence MUST be met by the platform manufacturer using some physical mechanism.

## 10.17 Other Specifications

### ***Start of informative comment:***

There are other security specifications and this section describes them and what level of compliance the TCPA may have with them.

- **Rainbow Series:** The Rainbow Series of specifications is being phased out by Protection Profiles. There is no requirement that the TCPA be Orange Book compatible.
- **ITSEC:** ITSEC is a European standard that is being phased out by Protection Profiles. There is no requirement that TCPA use any ITSEC specifications.
- **FIPS:** The FIPS 140 specification covers cryptographic modules and the hardware implementation of these modules. In many ways, Protection Profiles and FIPS overlap. Some of the FIPS 140 requirements are specified in this specification; however, compliance with the entire specification is not required.

### ***End of informative comment.***

Individual manufacturers MAY do the additional design and testing to obtain a FIPS 140 certification, but there is no requirement that a TCPA device obtain this testing.

Specifications or standards included in this specification

- **PKCS#1:** RSA Data Security, Inc. Public-Key Cryptography Standards (PKCS) Version 2.0
  - RSAES\_OAEP (2.0)
  - RSASSA-PKCS1-v1\_5
- **ITU-T Recommendation X.509 | ISO/IEC 9594-8:** "Information technology - Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks", 4<sup>th</sup> Edition.
- **DES/3DES:** Data Encryption Standard FIPS 46-3 (DES) : National Institute of Standards and Technology
- **ASN.1:** Abstract Syntax Notation One : ITU-T Recommendations X.680-X.683
- **FIPS 140-1:** Federal Information Processing Standards Publication 140-1 "Security Requirements for Cryptographic Modules"
- **BER:** Basic Encoding Rules : ITU-T Recommendation X.690-691 (1997)
- **ISO 15408** (Common Criteria)
- **SHA-1:** Secure Hash Algorithm : NIST FIPS PUB 180-1, "Secure Hash Standard," : National Institute of Standards and Technology
- **RFC 2104** (HMAC)

## 11. Appendix A: Glossary

### 3DES

DES using a key of a size that is 3X the size that of a DES key. See **DES**.

### Blob

Opaque data of fixed or variable size. The meaning and interpretation of the data is outside the scope and context of the Subsystem.

### Challenger

An entity that requests and has the ability to interpret integrity metrics from a Subsystem.

### Conformance Credential

A credential that states the conformance to the TCPA specification of: the TPM; the method of incorporation of the TPM into the platform; the RTM; and the method of incorporation of the RTM into the platform.

### Denial-of-service attack

A attack on a system (or subsystem) which has no affect on information except to prevent its use.

### DES

Symmetric key encryption using a key size of 56 bits defined by NIST as FIPS 46-3. Reference <http://csrc.ncsl.nist.gov/cryptval/des.htm>.

### Endorsement Credential

A credential containing a public key (the endorsement public key) that was generated by a genuine TPM.

### Endorsement Key

A term used ambiguously, depending on context, to mean a pair of keys, or the public key of that pair, or the private key of that pair; an asymmetric key pair generated by a TPM that is used as proof that a TPM is a genuine TPM; the public endorsement key (PUBEK); the private endorsement key (PRIVEK).

### Identity Credential

A credential issued by a Privacy CA that provides an identity for the TPM.

### Integrity metric(s)

Values that are the results of measurements on the integrity of the platform.

### Man-in-the-middle attack

An attack by an entity intercepting communications between two others without their knowledge and by intercepting that communication is able to obtain or modify the information between them.

### Migratable

A key which may be transported outside the specific TPM.

### Non-Migratable

A key which cannot be transported outside a specific TPM; a key that is (statistically) unique to a particular TPM.

### Non-Volatile

Storage location or memory that retain their values after power-off or a TPM\_Init function.

### Owner

The entity that owns the platform in which a TPM is installed. Since there is, by definition, a one-to-one relationship between the TPM and the platform, the Owner is also the Owner of the TPM. The Owner of

the platform is not necessarily the “user” of the platform (e.g., in a corporation, the Owner of the platform might be the IT department while the user is an employee.) The Owner has administration rights over the TPM.

**PKI Identity Protocol**

The protocol used to insert anonymous identities into the TPM.

**Platform Credential**

A credential that states that a specific platform contains a genuine TPCA Subsystem.

**POST**

POST refers to the Power On Self Test performed by a PC.

**Protection Profile**

A document that defines all attacks and how they are resisted by the TPM, the RTM, and the methods by which they are incorporated into the platform.

**Privacy CA**

An entity that issues an Identity Credential for a TPM based on trust in the entities that vouch for the TPM via the Endorsement Credential, the Conformance Credential, and the Platform Credential.

**Private Endorsement Key (PRIVEK)**

The private key of the key pair that proves that a TPM is a genuine TPM. The PRIVEK is (statistically) unique to only one TPM.

**Public Endorsement Key (PUBEK)**

A public key that proves that a TPM is a genuine TPM. The PUBEK is (statistically) unique to only one TPM.

**Random number generator (RNG)**

A pseudo-random number generator that must be initialized with unpredictable data and provides, “random” numbers on demand.

**Root of Trust for Measurement (RTM)**

The point from which all trust in the measurement process is predicated.

**Root of Trust for Reporting (RTR)**

The point from which all trust in reporting of measured information is predicated.

**Root of Trust for Storing (RTS)**

The point from which all trust in Protected Storage is predicated.

**RSA**

An (asymmetric) encryption method using two keys: a private key and a public key. Reference: <http://www.rsa.com> .

**SHA-1**

A NIST defined hashing algorithm producing a 160 bit result from an arbitrary sized source as specified in FIPS 180-1. Reference: <http://csrc.nist.gov/cryptval/shs.html>.

**Storage Root Key (SRK)**

The root key of a hierarchy of keys associated with a TPM; generated within a TPM; a non-migratable key.

**Subsystem**

The combination of the TSS and the TPM.

**Support Services (TSS)**

Services to support the TPM but which do not need the protection of the TPM. The same as **Trusted Platform Support Services**.

**TCPA-protected capability**

A function which is protected within the TPM, and has access to TPM secrets.

**TPM Identity**

One of the anonymous PKI identities belonging to a TPM; a TPM may have multiple identities.

**TPM POST**

TPM POST refers to the Power On Self Test performed by a TPM.

**Trusted Platform Agent (TPA)**

Trusted Platform Agent; the component within the platform that reports integrity metrics, logs, Validation Data, etc. to a Challenger; outside the scope of this specification.

**Trusted Platform Measurement Store (TPMS)**

Storage locations within the Subsystem, which contain unprotected logs of measurement process.

**Trusted Platform Module (TPM)**

The set of functions and data that are common to all types of platform, which must be trustworthy if the Subsystem is to be trustworthy; a logical definition in terms of protected capabilities and shielded locations.

**Trusted Platform Support Services (TSS)**

The set of functions and data that are common to all types of platform, which are not required to be trustworthy (and therefore do not need to be part of the TPM).

**User**

An entity that uses the platform in which a TPM is installed. The only rights that a User has over a TPM are the rights given to the User by the Owner. These rights are expressed in the form of authorization data, given by the Owner to the User, that permits access to entities protected by the TPM. The User of the platform is not necessarily the "owner" of the platform (e.g., in a corporation, the owner of the platform might be the IT department while the User is an employee). There can be multiple Users.

**Validation Credential**

A credential that states values of measurements that should be obtained when measuring a particular part of the platform when the part is functioning as expected.

**Validation Data**

Data inside a Validation Credential; the values that the integrity measurements should produce when the part of a platform described by the Validation Credential is working correctly.

**Validation Entity**

An entity that issues a Validation Certificate for a component; the manufacturer of that component; an agent of the manufacturer of that component.

**Volatile**

Storage locations or memory that are either set to a predefined value (e.g., zero) or have values that are undefined upon completion of a power-on or TPM\_Init function.