

Information technology — Trusted Platform Module —

Part 2: Design principles

ICS 35.040

National foreword

This British Standard is the UK implementation of ISO/IEC 11889-2:2009.

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

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

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

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

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 June 2009.

© BSI 2009

ISBN 978 0 580 68271 1

Amendments/corrigenda issued since publication

Date	Comments
31 August 2009	Correction to BS identifier

**Information technology — Trusted
Platform Module —**

**Part 2:
Design principles**

*Technologies de l'information — Module de plate-forme de confiance —
Partie 2: Principes de conception*



Reference number
ISO/IEC 11889-2:2009(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2009

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2009

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Table of Contents

1. Scope	1
1.1 Key words	1
1.2 Statement Type	1
2. Normative references	2
3. Abbreviated Terms	3
4. Conformance	5
4.1 Introduction	5
4.2 Threat	6
4.3 Protection of functions	6
4.4 Protection of information	6
4.5 Side effects	7
4.6 Exceptions and clarifications	7
5. TPM Architecture	8
5.1 Interoperability	8
5.2 Components	8
5.2.1 Input and Output	9
5.2.2 Cryptographic Co-Processor	9
5.2.3 Key Generation	11
5.2.4 HMAC Engine	12
5.2.5 Random Number Generator	13
5.2.6 SHA-1 Engine	15
5.2.7 Power Detection	16
5.2.8 Opt-In	16
5.2.9 Execution Engine	17
5.2.10 Non-Volatile Memory	17
5.3 Data Integrity Register (DIR)	18
5.4 Platform Configuration Register (PCR)	18
6. Endorsement Key Creation	20
6.1 Controlling Access to PRIVEK	21
6.2 Controlling Access to PUBEK	21
7. Attestation Identity Keys	22
8. TPM Ownership	23
8.1 Platform Ownership and Root of Trust for Storage	23
9. Authentication and Authorization Data	24
9.1 Dictionary Attack Considerations	25
10. TPM Operation	26
10.1 TPM Initialization & Operation State Flow	27
10.1.1 Initialization	27

10.2	Self-Test Modes	28
10.2.1	Operational Self-Test	29
10.3	Startup	32
10.4	Operational Mode	33
10.4.1	Enabling a TPM	34
10.4.2	Activating a TPM	35
10.4.3	Taking TPM Ownership	36
10.4.4	Transitioning Between Operational States	38
10.5	Clearing the TPM	38
11.	Physical Presence	40
12.	Root of Trust for Reporting (RTR)	42
12.1	Platform Identity	42
12.2	RTR to Platform Binding	43
12.3	Platform Identity and Privacy Considerations	43
12.4	Attestation Identity Keys	43
12.4.1	AIK Creation	44
12.4.2	AIK Storage	45
13.	Root of Trust for Storage (RTS)	46
13.1	Loading and Unloading Blobs	46
14.	Transport Sessions and Authorization Protocols	47
14.1	Authorization Session Setup	48
14.2	Parameter Declarations for OIAP and OSAP Examples	50
14.2.1	Object-Independent Authorization Protocol (OIAP)	52
14.2.2	Object-Specific Authorization Protocol (OSAP)	56
14.3	Authorization Session Handles	59
14.4	Authorization-Data Insertion Protocol (ADIP)	60
14.5	AuthData Change Protocol (ADCP)	64
14.6	Asymmetric Authorization Change Protocol (AACCP)	65
15.	ISO/IEC 19790 Evaluations	66
15.1	TPM Profile for successful ISO/IEC 19790 evaluation	66
16.	Maintenance	67
16.1	Field Upgrade	69
17.	Proof of Locality	70
18.	Monotonic Counter	71
19.	Transport Protection	74
19.1	Transport encryption and authorization	75
19.1.1	MGF1 parameters	77
19.1.2	HMAC calculation	78
19.1.3	Transport log creation	78
19.1.4	Additional Encryption Mechanisms	78

19.2	Transport Error Handling	79
19.3	Exclusive Transport Sessions	79
19.4	Transport Audit Handling	80
19.4.1	Auditing of wrapped commands	80
20.	Audit Commands	81
20.1	Audit Monotonic Counter	83
21.	Design Section on Time Stamping	84
21.1	Tick Components	84
21.2	Basic Tick Stamp	85
21.3	Associating a TCV with UTC	85
21.4	Additional Comments and Questions	87
22.	Context Management	89
23.	Eviction	91
24.	Session pool	92
25.	Initialization Operations	93
26.	HMAC digest rules	94
27.	Generic authorization session termination rules	95
28.	PCR Grand Unification Theory	96
28.1	Validate Key for use	98
29.	Non Volatile Storage	100
29.1	NV storage design principles	101
29.1.1	NV Storage use models	101
29.2	Use of NV storage during manufacturing	103
30.	Delegation Model	104
30.1	Table Requirements	104
30.2	How this works	105
30.3	Family Table	106
30.4	Delegate Table	107
30.5	Delegation Administration Control	108
30.5.1	Control in Phase 1	109
30.5.2	Control in Phase 2	110
30.5.3	Control in Phase 3	110
30.6	Family Verification	110
30.7	Use of commands for different states of TPM	112
30.8	Delegation Authorization Values	112
30.8.1	Using the authorization value	112
30.9	DSAP description	113
31.	Physical Presence	116
31.1	Use of Physical Presence	116
32.	TPM Internal Asymmetric Encryption	117

32.1.1	TPM_ES_RSAESOAEP_SHA1_MGF1	117
32.1.2	TPM_ES_RSAESPKCSV15	118
32.1.3	TPM_ES_SYM_CTR	118
32.1.4	TPM_ES_SYM_OFB	118
32.2	TPM Internal Digital Signatures	118
32.2.1	TPM_SS_RSASSAPKCS1v15_SHA1	119
32.2.2	TPM_SS_RSASSAPKCS1v15_DER	119
32.2.3	TPM_SS_RSASSAPKCS1v15_INFO	120
32.2.4	Use of Signature Schemes	120
33.	Key Usage Table	121
34.	Direct Anonymous Attestation	123
34.1	TPM_DAA_JOIN	123
34.2	TPM_DAA_Sign	124
34.3	DAA Command summary	125
34.3.1	TPM setup	125
34.3.2	JOIN	126
34.3.3	SIGN	129
35.	General Purpose IO	132
36.	Redirection	133
37.	Structure Versioning	134
38.	Certified Migration Key Type	135
38.1	Certified Migration Requirements	135
38.2	Key Creation	136
38.3	Migrate CMK to a MA	136
38.4	Migrate CMK to a MSA	136
39.	Revoke Trust	138
40.	Mandatory and Optional Functional Blocks	139
41.	1.1a and 1.2 Differences	142
42.	Bibliography	143

Foreword

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

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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

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

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

- *Part 1: Overview*
- *Part 2: Design principles*
- *Part 3: Structures*
- *Part 4: Commands*

Introduction

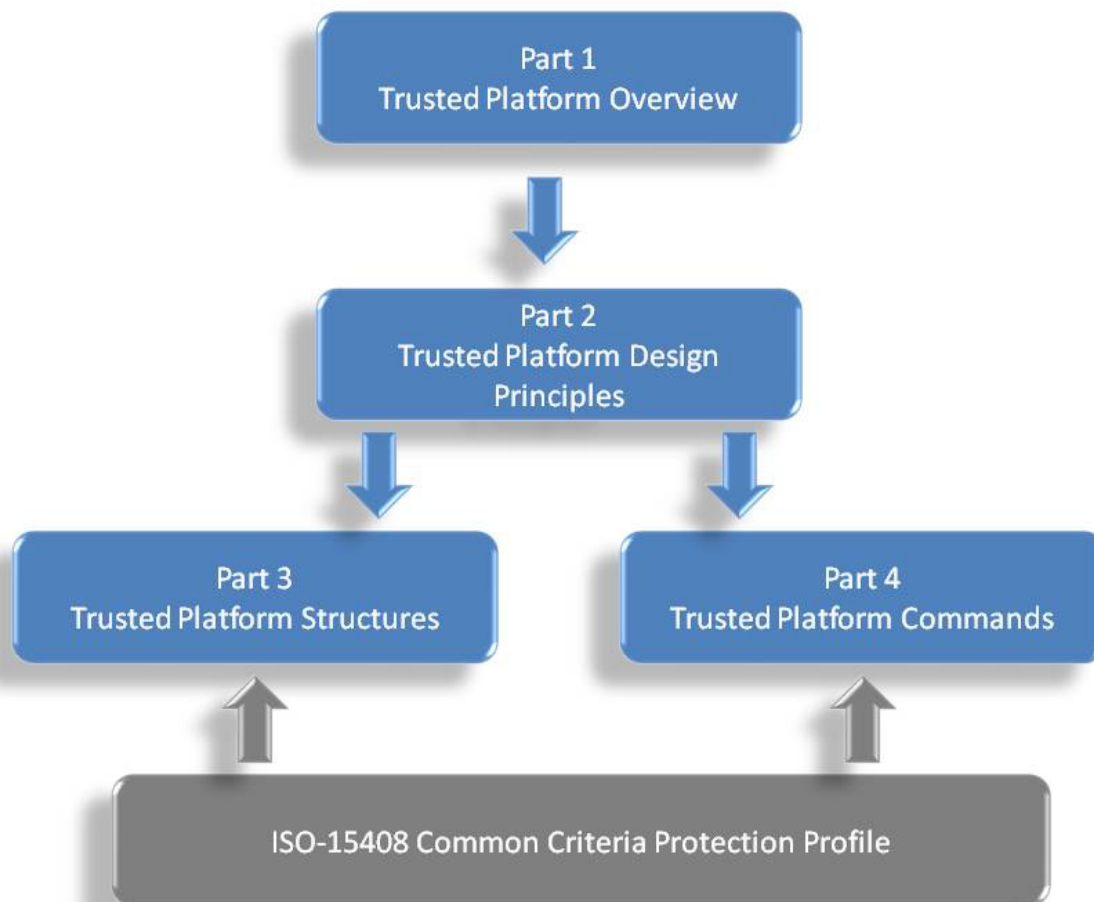


Figure 1. TPM Main Specification Roadmap

Start of informative comment

ISO/IEC 11889 is from the Trusted Computing Group (TCG) Trusted Platform Module (TPM) specification 1.2 version 103. The part numbers for ISO/IEC 11889 and the TCG specification do not match. The reason is the inclusion of the Overview document that is not a member of the TCG part numbering. The mapping between the two is as follows:

ISO Reference	TCG Reference
Part 1 Overview	Not published
Part 2 Design Principles	Part 1 Design Principles
Part 3 Structures	Part 2 Structures
Part 4 Commands	Part 3 Commands

End of informative comment

Information technology — Trusted Platform Module —

Part 2: Design principles

1. Scope

ISO/IEC 11889 defines the Trusted Platform Module (TPM), a device that enables trust in computing platforms in general. ISO/IEC 11889 is broken into parts to make the role of each document clear. Any version of the standard requires all parts to be a complete standard.

A TPM designer **MUST** be aware that for a complete definition of all requirements necessary to build a TPM, the designer **MUST** use the appropriate platform specific specification to understand all of the TPM requirements.

Part 2 defines the principles of TPM operation. The base operating modes, the algorithms and key choices, along with basic interoperability requirements make up the majority of the normative statements in part 2.

1.1 Key words

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

1.2 Statement Type

Please note a very important distinction between different sections of text throughout this document. 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.

For example:

Start of informative comment

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 standard the user must read the standard. (This use of **MUST** does not require any action).

End of informative comment

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

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

2. Normative references

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

ISO/IEC 8825-1|ITU-T X.690: Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)

ISO/IEC 10118-3, Information technology — Security techniques — Hash-functions — Part 3: Dedicated hash-functions, Clause 9, SHA-1

ISO/IEC 18033-3, Information technology — Security techniques — Encryption algorithms — Part 3, Block ciphers, Clause 5.1 AES

IEEE P1363, Institute of Electrical and Electronics Engineers: Standard Specifications For Public-Key Cryptography

IETF RFC 2104, Internet Engineering Task Force Request for Comments 2104: HMAC: Keyed-Hashing for Message Authentication

IETF RFC 2119, Internet Engineering Task Force Request for Comments 2119: Key words for use in RFCs to Indicate Requirement Levels

PKCS #1 Version 2.1, RSA Cryptography Standard. This document is superseded by P1363, except for section 7.2 that defines the V1.5 RSA signature scheme in use by the TPM.

3. Abbreviated Terms

Abbreviation	Description
AACP	Asymmetric Authorization Change Protocol
ADCP	Authorization Data Change Protocol
ADIP	Authorization Data Insertion Protocol
AIK	Attestation Identity Key
AMC	Audit Monotonic Counter
APIP	Time-Phased Implementation Plan
AuthData	Authentication Data or Authorization Data, depending on the context
BCD	Binary Coded Decimal
BIOS	Basic Input/Output System
CA	Certification of Authority
CDI	Controlled Data Item
CMK	Cerifiable/Certified Migratable Keys
CRT	Chinese Remainder Theorem
CRTM	Core Root of Trust Measurement
CTR	Counter-mode encryption
DAA	Direct Autonomous Attestation
DIR	Data Integrity Register
DOS	Disk Operating System
DSA	Digital Signature Algorithm
DSAP	Delegate-Specific Authorization Protocol
ECB	Electronic Codebook Mode
EK	Endorsement Key
ET	ExecuteTransport or Entity Type
FIPS	Federal Information Processing Standard
GPIO	General Purpose I/O
HMAC	Hash Message Authentication Code
HW	Hardware Interface
IB	Internal Base
I/O	Input/Output
IV	Initialization Vector
KH	Key Handle
LEAP	Lightweight Extensible Authentication Protocol for wireless computer networks
LK	Loaded Key
LOM	Limited Operation Mode
LPC	Low Pin Count
LSB	Least Significant Byte
MA	Migration Authority/Authorization
MIDL	Microsoft Interface Definition Language
MSA	Migration Selection Authority
MSB	Most Significant Byte
NV	Non-volatile

Abbreviation	Description
NVRAM	Non-Volatile Random Access Memory
OAEP	Optimal Asymmetric Encryption Padding
OEM	Original Equipment Manufacturer
OIAP	Object-Independent Authorization Protocol
OID	Object Identifier
OSAP	Object-Specific Authorization Protocol
PCR	Platform Configuration Register
PI	Personal Information
PII	Personally Identifiable Information
POST	Power On Self Test
PRIVEK	Private Endorsement Key
PRNG	Pseudo Random Number Generator
PSS	Probabilistic Signature Scheme
PUBEK	Public Endorsement Key
RNG	Random Number Generator
RSA	Algorithm for public-key cryptography. The letters R, S, and A represent the initials of the first public describers of the algorithm.
RTM	Release to Manufacturing/Ready to Market
RTR	Root of Trust for Reporting
RTS	Root of Trust for Storage
SHA	Secure Hash Algorithm
SRK	Storage Root Key
STF	Self Test Failed
TA	Time Authority
TBB	Threading Building Blocks
TCG	Trusted Computing Group
TCV	Tick Count Value
TIR	Tick Increment Rate
TIS	TPM Interface Specification
TNC	Trusted Network Connect
TOE	Target of Evaluation
TOS	Trusted Operating System
TPCA	Trusted Platform Computing Alliance
TPM	Trusted Platform Module
TPME	Trusted Platform Module Entity
TSC	Tick Stamp Counter
TSC_	TPM Software Connection, when used as a command prefix
TSN	Tick Session Name
TSR	Tick Stamp Reset
TSRB	TickStampReset for blob
TSS	TCG Software Stack
TTP	Trusted Third Party/Time-Triggered Protocol
TS	Tick Stamp
UTC	Universal Time Clock
VPN	Virtual Private Network

4. Conformance

4.1 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 TPM 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. The ordinal set defined in part II and III is the set of protected capabilities. The data structures in part II define the shielded locations.

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

ISO/IEC 11889 uses the concept of protected capabilities so as to distinguish platform capabilities that must be trustworthy. Trust in the TPM depends critically on the protected capabilities. Platform capabilities that are not protected capabilities must (of course) work properly if the TPM Subsystem is to function properly.

ISO/IEC 11889 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 TPM 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

1. The data structures described in ISO/IEC 11889-3 MUST NOT be instantiated in a TPM, except as data in TPM_Shielded-Locations.
2. The ordinal set defined in ISO/IEC 11889-3 and ISO/IEC 11889-4 MUST NOT be instantiated in a TPM, except as TPM_Protected-Capabilities.
3. Functions MUST NOT be instantiated in a TPM as TPM_Protected-Capabilities if they do not appear in the ordinal set defined in ISO/IEC 11889-3 or ISO/IEC 11889-4

4.2 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 TPM_Shielded-Locations and TPM_Protected-Capabilities in that platform.

The ISO/IEC 11889 standard defines the attacks that are resisted by the TPM. These attacks must be considered when determining whether the integrity of TPM_Protected-Capabilities and data in TPM_Shielded-Locations can be damaged. These attacks must be considered when determining whether there is a backdoor method of obtaining access to TPM_Protected-Capabilities and data in TPM_Shielded-Locations. These attacks must be considered when determining whether TPM_Protected-Capabilities have undesirable side effects.

End of informative comment

1. For the purposes of the “Protection” section of the standard, the threats that **MUST** be considered when determining whether the TPM facilitates subversion of TPM_Protected-Capabilities or data in TPM_Shielded-Locations **SHALL** include
 - a. The methods inherent in physical attacks that fail if the TPM complies with the “physical protection” requirements specified by ISO/IEC 11889
 - b. All methods that require execution of instructions in a computing engine in the platform

4.3 Protection of functions

Start of informative comment

A TPM_Protected-Capability must be used to modify TPM_Protected-Capabilities. Other methods must not be allowed to modify TPM_Protected-Capabilities. Otherwise, the integrity of TPM_Protected-Capabilities is unknown.

End of informative comment

1. A TPM **SHALL NOT** facilitate the alteration of TPM_Protected-Capabilities, except by TPM_Protected-Capabilities.

4.4 Protection of information

Start of informative comment

TPM_Protected-Capabilities must provide the only means from outside the TPM to access information represented by data in TPM_Shielded-Locations. Otherwise, a rogue can reveal data in TPM_Shielded-Locations, or create a derivative of data from TPM_Shielded-Locations (in a way that maintains some or all of the information content of the data) and reveal the derivative.

End of informative comment

1. A TPM **SHALL NOT** export data that is dependent upon data structures described in part 3 of ISO/IEC 11889, other than via a TPM_Protected-Capability.

4.5 Side effects

Start of informative comment

An implementation of a TPM_Protected-Capability must not disclose the contents of TPM_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

1. The implementation of a TPM_Protected-Capability in a TPM SHALL NOT facilitate the disclosure or the exposure of information represented by data in TPM-shielded-locations, except by means unavoidably inherent in the TPM definition.

4.6 Exceptions and clarifications

Start of informative comment

These exceptions to the blanket statements in the generic “protection” requirements (above) are fully compatible with the intended effect of those statements. These exceptions affect ISO/IEC 11889-data that is available as plain-text outside the TPM and ISO/IEC 11889-data that can be used without violating security or privacy. These exceptions are valuable because they approve use of TPM resources by vendor-specific commands in particular circumstances.

These clarifications to the blanket statements of the generic “protection” requirements (above) do not materially change the effect of those statements, but serve to approve specific legitimate interpretations of the requirements.

End of informative comment

1. A Shielded Location is a place (memory, register, etc.) where data is protected against interference and exposure, independent of its form
2. A TPM_Protected-Capability is an operation defined in and restricted to those identified in part 3 and 4 of ISO/IEC 11889
3. A vendor specific command or capability MAY use the standard ISO/IEC 11889 owner/operator authorization mechanism
4. A vendor specific command or capability MAY utilize a TPM_PUBKEY structure stored on the TPM so long as the usage of that TPM_PUBKEY structure is authorized using the standard ISO/IEC 11889 authorization mechanism.
5. A vendor specific command or capability MAY use a sequence of standard ISO/IEC 11889 commands. The command MUST propagate the locality used for the call to the used ISO/IEC 11889 commands or capabilities, or set locality to 0.
6. A vendor specific command or capability that takes advantage of exceptions and clarifications to the “protection” requirements MUST be defined as part of the security target of the TPM. Such a vendor specific command or capability MUST be evaluated to meet the Platform Specific TPM and System Security Targets.
7. If a TPM employs vendor-specific cipher-text that is protected against subversion to the same or greater extent as internal TPM-resources stored outside the TPM with ISO/IEC 11889-defined methods, that vendor-specific cipher-text does not necessarily require protection from physical attack. If a TPM location stores only vendor-specific cipher-text that does not require protection from physical attack, that location can be ignored when determining whether the TPM complies with the “physical protection” requirements specified by ISO/IEC 11889.

5. TPM Architecture

5.1 Interoperability

Start of informative comment

The TPM supports a minimum set of algorithms and operations to meet ISO/IEC 11889 standard.

Algorithms

RSA, normative statements in section 5.2.2.1

SHA-1, normative statements in section 5.2.6

HMAC, normative statements in section 5.2.4

The algorithms and protocols are the minimum that the TPM supports. Additional algorithms and protocols may be available to the TPM. All algorithms and protocols available in the TPM would be included in the TPM and platform credential.

The reason to specify these algorithms is two fold. The first is to know and understand the security properties of selected algorithms; identify appropriate key sizes and ensure appropriate use in protocols. The second reason is to define a base level of algorithms for interoperability.

End of informative comment

5.2 Components

Start of informative comment

The following is a block diagram Figure 5:a shows the major components of a TPM.

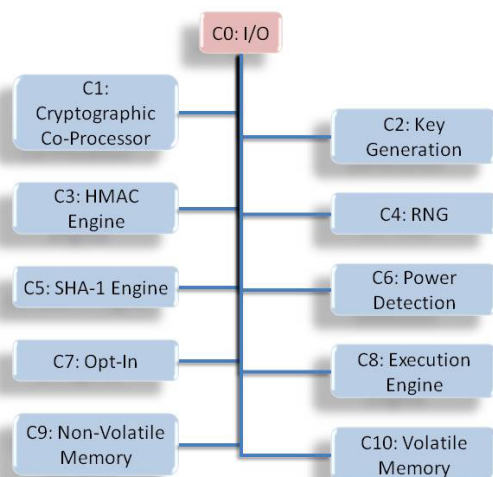


Figure 5:a - TPM Component Architecture

End of informative comment

5.2.1 Input and Output

Start of informative comment

The I/O component, Figure 5:a C0, manages information flow over the communications bus. It performs protocol encoding/decoding suitable for communication over external and internal buses. It routes messages to appropriate components. The I/O component enforces access policies associated with the Opt-In component as well as other TPM functions requiring access control.

ISO/IEC 11889 does not require a specific I/O bus. Issues around a particular I/O bus are the purview of a platform specific specification.

End of informative comment

1. The number of incoming operand parameter bytes must exactly match the requirements of the command ordinal. If the command contains more or fewer bytes than required, the TPM MUST return TPM_BAD_PARAMETER.

5.2.2 Cryptographic Co-Processor

Start of informative comment

The cryptographic co-processor, Figure 5:a C1, implements cryptographic operations within the TPM. The TPM employs conventional cryptographic operations in conventional ways. Those operations include the following:

Asymmetric key generation (RSA)

Asymmetric encryption/decryption (RSA)

Hashing (SHA-1)

Random number generation (RNG)

The TPM uses these capabilities to perform generation of random data, generation of asymmetric keys, signing and confidentiality of stored data.

The TPM may symmetric encryption for internal TPM use but does not expose any symmetric algorithm functions to general users of the TPM.

The TPM may implement additional asymmetric algorithms. TPM devices that implement different algorithms may have different algorithms perform the signing and wrapping.

If the TPM uses RSA with the required key length (2048 bits for storage keys), the output of all commands for key or data blob generation (e.g., TPM_CreateWrapKey, TPM_Seal, TPM_Sealx, TPM_MakeIdentity) consists of only one block. However, if the TPM uses other asymmetric algorithms that result in more than one output block for these commands, the integrity of the blobs must be protected by the TPM (by means of appropriate chaining mechanisms).

End of informative comment

1. The TPM MAY implement other asymmetric algorithms such as DSA or elliptic curve.
 - a. 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.
 - b. If the output key or data blob generated with a storage key consists of more than one block, the TPM MUST protect the integrity of the blob by means of appropriate chaining mechanisms.
2. All storage keys MUST be of strength equivalent to a 2048 bits RSA key or greater. The TPM SHALL NOT load a storage key whose strength less than that of a 2048 bits RSA key.
3. All AIK MUST be of strength equivalent to a 2048 bits RSA key, or greater.

5.2.2.1 RSA Engine

Start of informative comment

The RSA asymmetric algorithm is used for digital signatures and for encryption.

For RSA keys the P1363 standard provides the implementation details for digital signature, encryption and data formats.

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.

ISO/IEC 11889 retains support for 512-bit keys only for the purposes of backwards compatibility with legacy applications.

End of informative comment

1. The TPM MUST support RSA.
2. The TPM MUST use the RSA algorithm for encryption and digital signatures.
3. The TPM MUST support key sizes of 512, 1024, and 2048 bits. The TPM MAY support other key sizes.
 - a. The minimum RECOMMENDED key size is 2048 bits.
4. The RSA public exponent MUST be e , where $e = 2^{16} + 1$.
5. 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.

5.2.2.2 Signature Operations

Start of informative comment

The TPM performs signatures on both internal items and on requested external blobs. The rules for signatures apply to both operations.

End of informative comment

1. The TPM MUST use the RSA algorithm for signature operations where signed data is verified by entities other than the TPM that performed the sign operation.
2. The TPM MAY use other asymmetric algorithms for signatures; however, there is no requirement that other TPM devices either accept or verify those signatures.
3. The TPM MUST use P1363 for the format and design of the signature output.

5.2.2.3 Symmetric Encryption Engine

Start of informative comment

The TPM uses symmetric encryption to encrypt authentication information, provide confidentiality in transport sessions and provide internal encryption of blobs stored off the TPM.

For authentication and transport sessions, the mandatory mechanism is a pseudo one-time-pad with XOR. The mechanism to generate the one-time-pad is MGF1 and the nonces from the session protocol. When encrypting authorization data, the authorization data and the nonces are the same size, 20 bytes, so a direct XOR is possible.

For transport sessions the size of data is larger than the nonces so there needs to be a mechanism to expand the entropy to the size of the data. The mechanism to expand the entropy is the MGF1 function from P1363. This function provides a known mechanism that does not lower the entropy of the nonces.

AES may be supported as an alternate symmetric key encryption algorithm. The mode of use for AES in the TPM is counter mode and the actions for an operation define what the starting counter value is.

Internal protection of information can use any symmetric algorithm that the TPM designer feels provides the proper level of protection.

The TPM does not expose any of the symmetric operations for general message encryption.

End of informative comment

5.2.2.4 Using Keys

Start of Informative comments:

Keys can be symmetric or asymmetric.

As the TPM does not have an exposed symmetric algorithm, the TPM is only a generator, storage device and protector of symmetric keys. Generation of the symmetric key would use the TPM RNG. Storage and protection of the symmetric key would be provided by the BIND and SEAL capabilities of the TPM. If the caller wants to ensure that the release of a symmetric key is not exposed after UNBIND/UNSEAL on delivery to the caller, the caller should use a transport session with confidentiality set.

For asymmetric algorithms, the TPM generates and operates on RSA keys. The keys can be held only by the TPM or in conjunction with the caller of the TPM. If the private portion of a key is in use outside of the TPM it is the responsibility of the caller and user of that key to ensure the protections of the key.

The TPM has provisions to indicate if a key is held exclusively for the TPM or can be shared with entities off of the TPM.

End of informative comments.

1. A secret key is a key that is a private asymmetric key or a symmetric key.
2. Data SHOULD NOT be used as a secret key by a TPM_Protected-Capability unless that data has been extant only in a shielded location.
3. A key generated by a TPM_Protected-Capability SHALL NOT be used as a secret key unless that key has been extant only in a shielded location.
4. A secret key obtained by a TPM_Protected-Capability from a Protected Storage blob SHALL be extant only in a shielded location.

5.2.3 Key Generation

Start of informative comment

The Key Generation component, Figure 5:a C2, creates RSA key pairs and symmetric keys. ISO/IEC 11889 places no minimum requirements on key generation times for asymmetric or symmetric keys.

End of informative comment

5.2.3.1 Asymmetric – RSA

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).

5.2.3.2 Nonce Creation

The creation of all nonce values MUST use the next n bits from the TPM RNG.

5.2.4 HMAC Engine

Start of informative comment

The HMAC engine, Figure 5:a C3, provides two pieces of information to the TPM: proof of knowledge of the AuthData 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. ISO/IEC 11889 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

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

where

H = the SHA1 hash operation

K = the key or the AuthData

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

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. Each command interface definition indicates what parameters are involved in the HMAC calculation.

5.2.5 Random Number Generator

Start of informative comment

The Random Number Generator (RNG) component, Figure 6:a C4 is the source of randomness in the TPM. The TPM uses these random values for nonces, key generation, and randomness in signatures.

The RNG consists of a state-machine that accepts and mixes unpredictable data and a post-processor that has a one-way function (e.g. SHA-1). The idea behind the design is that a TPM can be good source of randomness without having to require a genuine source of hardware entropy.

The state-machine can have a non-volatile state initialized with unpredictable random data during TPM manufacturing before delivery of the TPM to the customers. The state-machine can accept, at any time, further (unpredictable) data, or entropy, to salt the random number. Such data comes from hardware or software sources – for example; from thermal noise, or by monitoring random keyboard strokes or mouse movements. The RNG requires a reseeding after each reset of the TPM. A true hardware source of entropy is likely to supply entropy at a higher baud rate than a software source.

When adding entropy to the state-machine, the process must ensure that after the addition, no outside source can gain any visibility into the new 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 after shipment of the TPM. The RNG post-processor condenses the output of the state-machine into data that has sufficient and uniform entropy. The one-way function should use more bits of input data than it produces as output.

Our definition of the RNG allows implementation of a Pseudo Random Number Generator (PRNG) algorithm. However, on devices where a hardware source of entropy is available, a PRNG need not be implemented. ISO/IEC 11889 refers to both RNG and PRNG implementations as the RNG mechanism. From a user standpoint, the difference between a RNG and a PRNG is nonexistent. However, from a designer standpoint the difference between an RNG and PRNG will dictate various implementation choices. The ISO/IEC 11889 specification does not distinguish between the two types as the specification does not deal with implementation issues.

The TPM should be able to provide 32 bytes of randomness on each call. Larger requests may fail with not enough randomness being available.

End of informative comment

1. The RNG for the TPM will consist of the following components:
 - a. Entropy source and collector
 - b. State register
 - c. Mixing function
2. The RNG capability is a TPM_Protected-Capability with no access control.
3. The RNG output may or may not be shielded data. When the data is for internal use by the TPM (e.g., generation of tpmProof or an asymmetric key), the data **MUST** be held in a shielded location. The RNG output for internal use **MUST** not be known outside the TPM. In particular, it **MUST** not be known by the TPM manufacturer. When the data is for use by the TSS or another external caller, the data is not shielded.

5.2.5.1 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 collector differs from the mixing function in 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

1. The entropy source **MUST** provide entropy to the state register in a manner that provides entropy that is not visible to an outside process.
 - a. For compliance purposes, the entropy source **MAY** be outside of the TPM; however, attention **MUST** be paid to the reporting mechanism.
2. The entropy source **MUST** provide the information only to the state register.
 - a. 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.
 - b. 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.

5.2.5.2 State Register

Start of informative comment

The state register implementation may use two registers: a non-volatile register `rngState` 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;

To avoid overuse of the flash, as the number of writes to a flash device are limited.

End of informative comment

1. The state register is in a TPM shielded-location.
 - a. The state register **MUST** be non-volatile.
 - b. The update function to the state register is a TPM protected-capability.
 - c. The primary input to the update function **SHOULD** be the entropy collector.
2. 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.
 - a. This requirement implies that the addition of known data **MUST NOT** result in a decrease in the entropy of the state register.
3. The TPM **MUST NOT** export the state register.

5.2.5.3 Mixing Function

Start of informative comment

The mixing function takes the state register and produces output. The mixing function is a TPM protected-capability. The mixing function takes the value from a state register and creates the RNG output. If the entropy source has a bias, then the collector takes that bias into account before sending the information to the state register.

End of informative comment

1. Each use of the mixing function **MUST** affect the state register.
 - a. This requirement is to affect the volatile register and does not need to affect the non-volatile state register.

5.2.5.4 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

1. The RNG **MUST NOT** output any bits after a system reset until the following occurs:
 - a. 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.
 - b. 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.

5.2.6 SHA-1 Engine

Start of informative comment

The SHA-1, Figure 5:a C5, hash capability is primarily used by the TPM, as it is a trusted implementation of a hash algorithm. The hash interfaces are exposed outside the TPM to support Measurement taking during platform boot phases and to allow environments that have limited capabilities access to a hash functions. The TPM is not a cryptographic accelerator. ISO/IEC 11889 does not specify minimum throughput requirements for TPM hash services.

End of informative comment

1. The TPM **MUST** implement the SHA-1 hash algorithm as defined by ISO/IEC 10118-3, Clause 9.
2. The output of SHA-1 is 160 bits and all areas that expect a hash value are **REQUIRED** to support the full 160 bits.
3. The only commands that **SHALL** be presented to the TPM in-between a TPM_SHA1Start command and a TPM_SHA1Complete command **SHALL** be a variable number (possibly 0) of TPM_SHA1Update commands.
 - a. The TPM_SHA1Update commands can occur in a transport session.
4. Throughout all parts of the specification the characters $x1 \parallel x2$ imply the concatenation of $x1$ and $x2$

5.2.7 Power Detection

Start of informative comment

The power detection component, Figure 5:a C6, manages the TPM power states in conjunction with platform power states. ISO/IEC 11889 requires that the TPM be notified of all power state changes.

Power detection also supports physical presence assertions. The TPM may restrict command-execution during periods when the operation of the platform is physically constrained. In a PC, operational constraints occur during the power-on self-test (POST) and require Operator input via the keyboard. The TPM might allow access to certain commands while in a constrained execution mode or boot state. At some critical point in the POST process, the TPM may be notified of state changes that affect TPM command processing modes.

End of informative comment

5.2.8 Opt-In

Start of informative comment

The Opt-In component, Figure 5:a C7, provides mechanisms and protections to allow the TPM to be turned on/off, enabled/disabled, activated/deactivated. The Opt-In component maintains the state of persistent and volatile flags and enforces the semantics associated with these flags.

The setting of flags requires either authorization by the TPM Owner or the assertion of physical presence at the platform. The platform's manufacturer determines the techniques used to represent physical-presence. The guiding principle is that no remote entity should be able to change TPM status without either knowledge of the TPM Owner or the Operator is physically present at the platform. Physical presence may be asserted during a period when platform operation is constrained such as power-up.

Non-Volatile Flags:

PhysicalPresenceLifetimeLock

PhysicalPresenceHWEEnable

PhysicalPresenceCMDEnable

Volatile Flags:

PhysicalPresenceV

The following truth table explains the conditions in which the PhysicalPresenceV flag may be altered:

Persistent / Volatile	P	P	P	V	
Control Flags	PhysicalPresenceLifetimeLock	PhysicalPresenceHWEEnable	PhysicalPresenceCMDEnable	PhysicalPresenceV	
Volatile Access Semantics to Physical Presence Flag	-	F	F	-	No access to PhysicalPresenceV flag.
	-	F	T	T	
	-	-	T	F	Access to PhysicalPresenceV flag through TCS_PhysicalPresence command enabled.
	-	T	-	-	Access to PhysicalPresenceV flag through hardware signal enabled.
	-	T	T	F	Access to PhysicalPresenceV flag through hardware signal or TCS_PhysicalPresence command enabled.

Persistent / Volatile	P	P	P	V	
Control Flags	PhysicalPresenceLifetimeLock	PhysicalPresenceHWEEnable	PhysicalPresenceCMDEnable	PhysicalPresenceV	
Persistent Access Semantics to Physical Presence Flag	T	F	F	-	Access to PhysicalPresenceV flag permanently disabled.
	T	F	T	T	
	T	F	T	F	Exclusive access to PhysicalPresenceV flag through TCS_PhysicalPresence command permanently enabled.
	T	T	F	-	Exclusive access to PhysicalPresenceV flag through hardware signal permanently enabled.
	T	T	T	F	Access to PhysicalPresenceV flag through hardware signal or TCS_PhysicalPresence command permanently enabled.

Table 5:a - Physical Presence Semantics

ISO/IEC 11889 also recognizes the concept of unambiguous physical presence. Conceptually, the use of dedicated electrical hardware providing a trusted path to the Operator has higher precedence than the physicalPresenceV flag value. Unambiguous physical presence may be used to override physicalPresenceV flag value under conditions specified by platform specific design considerations.

Additional details relating to physical presence can be found in sections on Volatile and Non-volatile memory.

End of informative comment

5.2.9 Execution Engine

Start of informative comment

The execution engine, Figure 5:a C8, runs program code to execute the TPM commands received from the I/O port. The execution engine is a vital component in ensuring that operations are properly segregated and shield locations are protected.

End of informative comment

5.2.10 Non-Volatile Memory

Start of informative comment

Non-volatile memory component, Figure 5:a C9, is used to store persistent identity and state associated with the TPM. The NV area has set items (like the EK) and also is available for allocation and use by entities authorized by the TPM Owner.

The TPM designer should consider the use model of the TPM and if the use of NV storage is a concern. NV storage does have a limited life and using the NV storage in a high volume use model may prematurely wear out the TPM.

End of informative comment

5.3 Data Integrity Register (DIR)

Start of informative comment

The DIR were a version 1.1 function. They provided a place to store information using the TPM NV storage.

In 1.2 the DIR are deprecated and the use of the DIR should move to the general purpose NV storage area.

The TPM must still support the functionality of the DIR register in the NV storage area.

End of informative comment

1. A TPM MUST provide one Data Integrity Register (DIR)
 - a. The TPM DIR commands are deprecated in 1.2
 - b. The TPM MUST reserve the space for one DIR in the NV storage area
 - c. The TPM MAY have more than 1 DIR.
2. The DIR MUST be 160-bit values and MUST be held in TPM shielded-locations.
3. The DIR MUST be non-volatile (values are maintained during the power-off state).
 - a. A TPM implementation need not provide the same number of DIRs as PCRs.

5.4 Platform Configuration Register (PCR)

Start of informative comment

A Platform Configuration Register (PCR) is a 160-bit storage location for discrete integrity measurements. There are a minimum of 16 PCR registers. All PCR registers are shielded-locations and are inside of the TPM. The decision of whether a PCR contains a standard measurement or if the PCR is available for general use is deferred to the platform specific specification.

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 PCR is designed to hold an unlimited number of measurements in the register. It does this by using a cryptographic hash and hashing all updates to a PCR. The pseudo code for this is:

$$\text{PCR}_i \text{ New} = \text{HASH} (\text{PCR}_i \text{ Old value} \parallel \text{value to add})$$

There are two salient properties of cryptographic hash that relate to PCR construction. Ordering – meaning updates to PCRs are not commutative. For example, measuring (A then B) is not the same as measuring (B then A).

The other hash property is one-way-ness. This property means it should be computationally infeasible for an attacker to determine the input message given a PCR value. Furthermore, subsequent updates to a PCR cannot be determined without knowledge of the previous PCR values or all previous input messages provided to a PCR register since the last reset.

End of informative comment

1. The PCR MUST be a 160-bit field that holds a cumulatively updated hash value
2. The PCR MUST have a status field associated with it
3. The PCR MUST be in the RTS and should be in volatile storage
4. The PCR MUST allow for an unlimited number of measurements to be stored in the PCR
5. The PCR MUST preserve the ordering of measurements presented to it
6. A PCR MUST be set to the default value as specified by the PCRReset attribute
7. A TPM implementation MUST provide 16 or more independent PCRs. These PCRs are identified by index and MUST be numbered from 0 (that is, PCR0 through PCR15 are required for ISO/IEC 11889 compliance). Vendors MAY implement more registers for general-purpose use. Extra registers MUST be numbered contiguously from 16 up to max – 1, where max is the maximum offered by the TPM.
8. The ISO/IEC 11889-protected capabilities that expose and modify the PCRs use a 32-bit index, indicating the maximum usable PCR index. However, ISO/IEC 11889 reserves register indices 230 and higher for later versions of the specification. A TPM implementation MUST NOT provide registers with indices greater than or equal to 230. In ISO/IEC 11889, the following terminology is used (although this internal format is not mandated).
9. The PSS MUST define at least define one measurement that the RTM MUST make and the PCR where the measurement is stored.
10. A ISO/IEC 11889 measurement agent MAY discard a duplicate event instead of incorporating it in a PCR, provided that:
11. A relevant ISO/IEC 11889 platform specification explicitly permits duplicates of this type of event to be discarded
12. The PCR already incorporates at least one event of this type
13. An event of this type previously incorporated into the PCR included a statement that duplicate such events may be discarded. This option could be used where frequent recording of sleep states will adversely affect the lifetime of a TPM, for example.
14. 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 their default value (either 0x00...00 or 0xFF...FF). Additionally, the UINT32 flags MUST be set to zero.

6. Endorsement Key Creation

Start of informative comment

The TPM contains a 2048-bit RSA key pair called the endorsement key (EK). The public portion of the key is the PUBEK and the private portion the PRIVEK. Due to the nature of this key pair, both the PUBEK and the PRIVEK have privacy and security concerns.

The TPM has the EK generated before the end customer receives the platform. The Trusted Platform Module Entity (TPME) that causes EK generation is also the entity that will create and sign the EK credential attesting to the validity of the TPM and the EK. The TPME is typically the TPM manufacturer.

The TPM can generate the EK internally using the TPM_CreateEndorsementKey or by using an outside key generator. The EK needs to indicate the genealogy of the EK generation.

Subsequent attempts to either generate an EK or insert an EK must fail.

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.

End of informative comment

1. The EK MUST be a 2048-bit RSA key
 - a. The public portion of the key is the PUBEK
 - b. The private portion of the key is the PRIVEK
 - c. The PRIVEK SHALL exist only in a TPM_Shielded-Location.
2. Access to the PRIVEK and PUBEK MUST only be via TPM_Protected-Capabilities
 - a. The protected capabilities MUST require TPM_Owner-Authentication or operator physical presence
3. The generation of the EK may use a process external to the TPM and TPM_CreateEndorsementKeyPair
 - a. The external generation MUST result in an EK that has the same properties as an internally generated EK
 - b. The external generation process MUST protect the EK from exposure during the generation and insertion of the EK
 - c. After insertion of the EK the TPM state MUST be the same as the result of the TPM_CreateEndorsementKeyPair execution
 - d. The process MUST guarantee correct generation, cryptographic strength, uniqueness, privacy, and installation into a genuine TPM, of the EK
 - e. The entity that signs the EK credential MUST be satisfied that the generation process properly generated the EK and inserted it into the TPM
 - f. The process MUST be defined in the target of evaluation (TOE) of the security target in use to evaluate the TPM

6.1 Controlling Access to PRIVEK

Start of informative comment

Exposure of the PRIVEK is a security concern.

The TPM must ensure that the PRIVEK is not exposed outside of the TPM

End of informative comment

1. The PRIVEK MUST never be out of the control of a TPM_Shielded-Location

6.2 Controlling Access to PUBEK

Start of informative comment

There are no security concerns with exposure or use of the PUBEK.

Privacy guidelines suggest that PUBEK could be considered personally identifiable information (PII) if it were associated in some way with personal information (PI) or associated with other PII, but PUBEK alone cannot be considered PII. Arbitrary random numbers do not represent a threat to privacy unless further associated with PI or PII. The PUBEK is an arbitrary random number that may be associated with aggregate platform information, but not personally identifiable information.

An EK may become associated with personally identifiable information when an alias platform identifier (AIK) is also associated with PI. The attestation service could include personal information in the AIK credential, thereby making the AIK-PUBEK association PII – but not before.

The association of PUBEK with AIK therefore is important to protect via privacy guidelines. The owner/user of the TPM should be able to control whether PUBEK is disclosed along with AIK. The owner/user should be notified of personal information that might be added to an AIK credential, which could result in AIK being considered PII. The owner/user should be able to evaluate the mechanisms used by an attestation entity to protect PUBEK-AIK associations before disclosure occurs. No other entity should be privy to owner/user authorized disclosure besides the intended attestation entity.

Several commands may be used to negotiate the conditions of PUBEK-AIK disclosure. TPM_MakeIdentity discloses PUBEK-AIK in the context of requesting an AIK credential. TPM_ActivateIdentity ensures the owner/user has not been spoofed by an interloper. These interfaces allow the owner/user to choose whether disclosure is acceptable and control the circumstances under which disclosure takes place. They do not allow the owner/user the ability to retain control of PUBEK-AIK subsequent to disclosure except by traditional means of trusting the attestation entity to abide by an acceptable privacy policy. The owner/user is able to associate the accepted privacy policy with the disclosure operation (e.g. TPM_MakeIdentity).

A persistent flag called readPubek can be set to TRUE to permit reading of PUBEK via TPM_ReadPubek. Reporting the PUBEK value is not considered privacy sensitive because it cannot be associated with any of the AIK keys managed by the TPM without using TPM protected-capabilities. Keys are encrypted with a nonce when flushed from TPM shielded-locations. Cryptanalysis of flushed keys will not reveal an association of EK to any AIK.

The command that manipulates the readPubek flag is TPM_DisablePubekRead.

End of informative comment

7. Attestation Identity Keys

Start of informative comment

See 12.4 Attestation Identity Keys.

End of informative comment

8. TPM Ownership

Start of informative comment

Taking ownership of a TPM is the process of inserting a shared secret into a TPM shielded-location. Any entity that knows the shared secret is a TPM Owner. Proof of ownership occurs when an entity, in response to a challenge, proves knowledge of the shared secret. Certain operations in the TPM require authentication from a TPM Owner.

Certain operations also allow the human, with physical possession of the platform, to assert TPM Ownership rights. When asserting TPM Ownership, using physical presence, the operations must not expose any secrets protected by the TPM.

The platform owner controls insertion of the shared secret into the TPM. The platform owner sets the NV persistent flag ownershipEnabled that allows the execution of the TPM_TakeOwnership command. The TPM_SetOwnerInstall, the command that controls the value ownershipEnabled, requires the assertion of physical presence.

Attempting to execute TPM_TakeOwnership fails when a TPM already has an owner. To remove an owner when the current TPM Owner is unable to remove themselves, the human that is in possession of the platform asserts physical presence and executes TPM_ForceClear which removes the shared secret.

The insertion protocol that supplies the shared secret has the following requirements: confidentiality, integrity, remoteness and verifiability.

To provide confidentiality the proposed TPM Owner encrypts the shared secret using the PUBEK. This requires the PRIVEK to decrypt the value. As the PRIVEK is only available in the TPM the encrypted shared secret is only available to the intended TPM.

The integrity of the process occurs by the TPM providing proof of the value of the shared secret inserted into the TPM.

By using the confidentiality and integrity, the protocol is useable by TPM Owners that are remote to the platform.

The new TPM Owner validates the insertion of the shared secret by using integrity response.

End of informative comment

The TPM MUST ship with no Owner installed. The TPM MUST use the ownership-control protocol (OIAP or OSAP)

8.1 Platform Ownership and Root of Trust for Storage

Start of informative comment

The semantics of platform ownership are tied to the Root-of-trust-for-storage (RTS). The TPM_TakeOwnership command creates a new Storage Root Key (SRK) and new tpmProof value whenever a new owner is established. It follows that objects owned by a previous owner will not be inherited by the new owner. Objects that should be inherited must be transferred by deliberate data migration actions.

End of informative comment

9. Authentication and Authorization Data

Start of informative comment

Using security vernacular the terms below apply to the TPM for this discussion:

Authentication: The process of providing proof of claimed ownership of an object or a subject's claimed identity.

Authorization: Granting a subject appropriate access to an object.

Each TPM object that does not allow "public" access contains a 160-bit shared secret. This shared secret is enveloped within the object itself. The TPM grants use of TPM objects based on the presentation of the matching 160-bits using protocols designed to provide protection of the shared secret. This shared secret is called the AuthData.

Neither the TPM, nor its objects (such as keys), contain access controls for its objects (the exception to this is what is provided by the delegation mechanism). If an subject presents the AuthData, that subject is granted full use of the object based on the object's capabilities, not a set of rights or permissions of the subject. This apparent overloading of the concepts of authentication and authorization has caused some confusion. This is caused by having two similarly rooted but distinct perspectives.

From the perspective of the TPM looking out, this AuthData is its sole mechanism for authenticating the owner of its objects, thus from its perspective it is authentication data. However, from the application's perspective this data is typically the result of other functions that might perform authentications or authorizations of subjects using higher level mechanisms such as OS login, file system access, etc. Here, AuthData is a result of these functions so in this usage, it authorizes access to the TPM's objects. From this perspective, i.e., the application looking in on the TPM and its objects, the AuthData is authorization data. For this reason, and thanks to a common root within the English language, the term for this data is chosen to be AuthData and is to be interpreted or expanded as either authentication data or authorization data depending on context and perspective.

The term AuthData refers to the 160-bit value used to either prove ownership of, or authorization to use, an object. This is also called the object's shared secret. The term authorization will be used when referring the combined action of verifying the AuthData and allowing access to the object or function. The term authorization session applies to a state where the AuthData has been authentication and a session handle established that is associated with that authentication.

A wide-range of objects use AuthData. It is used to establish platform ownership, key use restrictions, object migration and to apply access control to opaque objects protected by the TPM.

AuthData is a 160-bit shared-secret. The assumption is the shared-secret and some randomness are mixed using SHA-1 digesting, but no specific function for generating AuthData is specified by ISO/IEC 11889.

ISO/IEC 11889 command processing sessions (e.g. OSAP, ADIP) may use AuthData as an initialization vector when creating a one-time pad. Session encryption is used to encrypt portions of command messages exchanged between TPM and a caller.

The TPM stores AuthData with TPM controlled-objects and in shielded-locations. AuthData is never in the clear, when managed by the TPM except in shielded-locations. Only TPM protected-capabilities may access AuthData (contained in the TPM). AuthData objects may not be used for any other purpose besides authentication and authorization of TPM operations on controlled-objects.

Outside the TPM, a reference monitor of some kind is responsible for protecting AuthData. AuthData should be regarded as a controlled data item (CDI) in the context of the security model governing the reference monitor. ISO/IEC 11889 expects this entity to preserve the interests of the platform Owner.

There is no requirement that instances of AuthData be unique.

End of informative comment

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

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

9.1 Dictionary Attack Considerations

Start of informative comment

The decision to provide protections against dictionary attacks is due to the inability of the TPM to guarantee that an authorization value has high entropy. While the creation and authorization protocols could change to support the assurance of high entropy values, the changes would be drastic and would totally invalidate any 1.x TPM version.

Version 1.1 explicitly avoided any requirements for dictionary attack mitigation.

Version 1.2 adds the requirement that the TPM vendor provide some assistance against dictionary attacks. The internal mechanism is vendor specific. The TPM designer should review the requirements for dictionary attack mitigation in the Common Criteria.

The 1.2 specification does not provide any functions to turn on the dictionary attack prevention. The specification does provide a way to reset from the TPM response to an attack.

To avoid all potential dictionary attacks the AuthData should be of high-entropy. If the AuthData is of low entropy, version 1.2 provides limited protection to the AuthData (see [ChenRyan08] L. Chen and M. D. Ryan, A solution for TCG TPM resistance to offline dictionary attacks on weak authorization data. Presented at *Future of Trust in Computing*, Berlin, 30 June – 2 July 2008]). The limited protection is the protection from online dictionary attacks described in this section.

By way of example, the following is a way to implement the dictionary attack mitigation.

The TPM keeps a count of failed authorization attempts. The vendor allows the TPM Owner to set a threshold of failed authorizations. When the count exceeds the threshold, the TPM locks up and does not respond to any requests for a time out period. The time out period doubles each time the count exceeds the threshold. If the TPM resets during a time out period, the time out period starts over after TPM_Init, or TPM_Startup. To reset the count and the time out period the TPM Owner executes TPM_ResetLockValue. If the authorization for TPM_ResetLockValue fails, the TPM must lock up for the entire time out period and no additional attempts at unlocking will be successful. Executing TPM_ResetLockValue when outside of a time out period still results in the resetting of the count and time out period.

End of informative comment

The TPM SHALL incorporate mechanism(s) that will provide some protection against exhaustive or dictionary attacks on the authorization values stored within the TPM.

This version of the TPM specification does NOT specify the particular strategy to be used. Some examples might include locking out the TPM after a certain number of failures, forcing a reboot under some combination of failures, or requiring specific actions on the part of some actors after an attack has been detected. The mechanisms to manage these strategies are vendor specific at this time.

If the TPM in response to the attacks locks up for some time period or requires a special operation to restart, the TPM MUST prevent any authorized TPM command and MAY prevent any TPM command from executing until the mitigation mechanism completes. The TPM Owner can reset the mechanism using the TPM_ResetLockValue command. TPM_ResetLockValue MUST be allowed to run exactly once while the TPM is locked up.

10. TPM Operation

Start of informative comment

Through the course of TPM operation, it may enter several operational modes that include power-up, self-test, administrative modes and full operation. This section describes TPM operational states and state transition criteria. Where applicable, the TPM commands used to facilitate state transition or function are included in diagrams and descriptions.

The TPM keeps the information relative to the TPM operational state in a combination of persistent and volatile flags. For ease of reading the persistent flags are prefixed by pFlags and the volatile flags prefixed by vFlags.

The following state diagram describes TPM operational states at a high level. Subsequent state diagrams drill-down to finer detail that describes fundamental operations, protections on operations and the transitions between them.

The state diagrams use the following notation:



- Signifies a state.



- Transitions between states are represented as a single headed arrows.



- Circular transitions indicate operations that don't result in a transition to another state.



- Decision boxes split state flow based on a logical test. Decision conditions are called Guards and are identified by bracketed text.

< [text] > Bracketed text indicates transitions that are gated. Text within the brackets describes the pre-condition that must be met before state transition may occur.

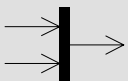
< /name > Transitions may list the events that trigger state transition. The forward slash demarcates event names.



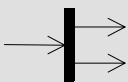
- The starting point for reading state diagrams.



- The ending point for state diagrams. Perpetual state systems may not have an ending indicator.



- The collection bar consolidates multiple identical transition events into a single transition arrow.



- The distribution bar splits transitions to flow into multiple states.



- The history indicator means state values are remembered across context switches or power-cycles.

End of informative comment

10.1 TPM Initialization & Operation State Flow

Start of informative comment

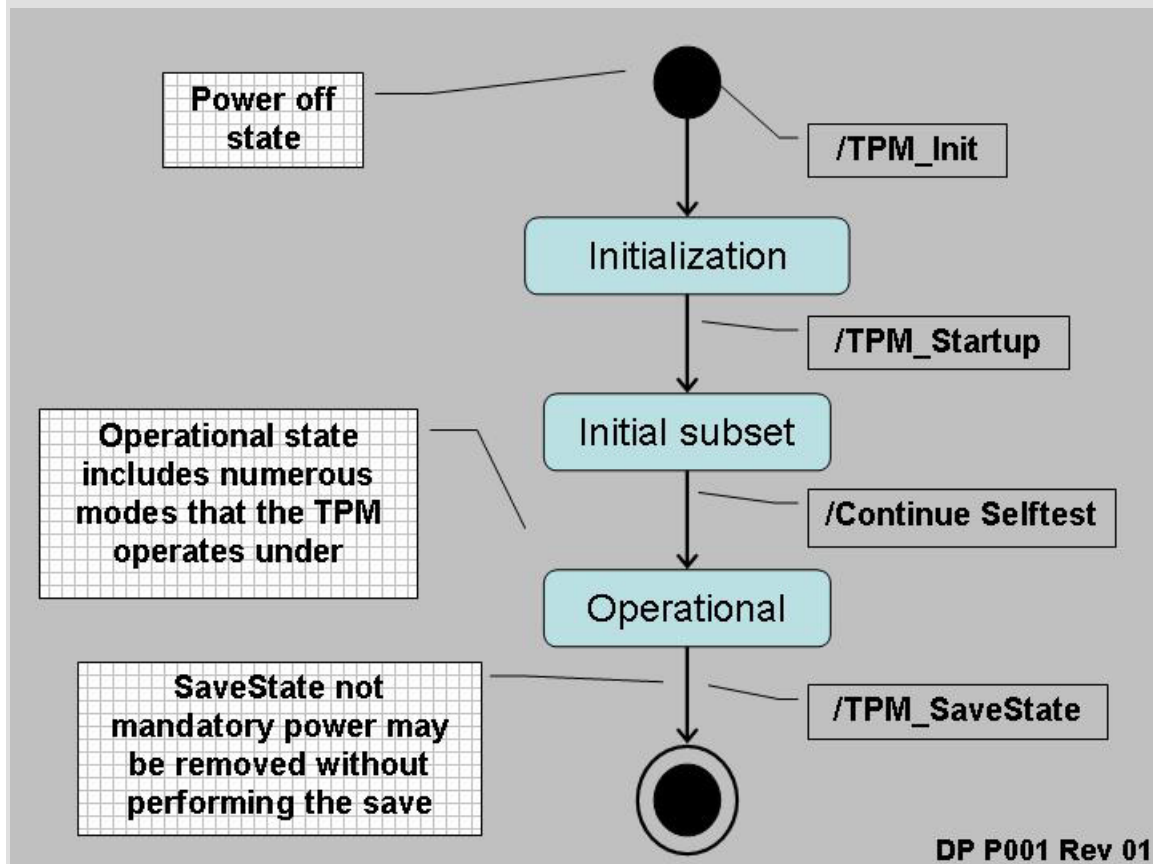


Figure 10:a - TPM Operational States

End of informative comment

10.1.1 Initialization

Start of informative comment

TPM_Init transitions the TPM from a power-off state to one where the TPM begins an initialization process. TPM_Init could be the result of power being applied to the platform or a hard reset.

TPM_Init sets an internal flag to indicate that the TPM is undergoing initialization. The TPM must complete initialization before it is operational. The completion of initialization requires the receipt of the TPM_Startup command.

The TPM is not fully operational until all of the self-tests are complete. Successful completion of the self-tests allows the TPM to enter fully operational mode.

Fully operational does not imply that all functions of the TPM are available. The TPM needs to have a TPM Owner and be enabled for all functions to be available.

The TPM transitions out of the operational mode by having power removed from the system. Prior to the exiting operational mode, the TPM prepares for the transition by executing the TPM_SaveState command. There is no requirement that TPM_SaveState execute before the transition to power-off mode occurs.

End of informative comment

1. After TPM_Init and until receipt of TPM_Startup the TPM MUST return TPM_INVALID_POSTINIT for all commands. Prior to receipt of TPM_Startup the TPM MAY enter shutdown or failure mode.

10.2 Self-Test Modes

Start of informative comment

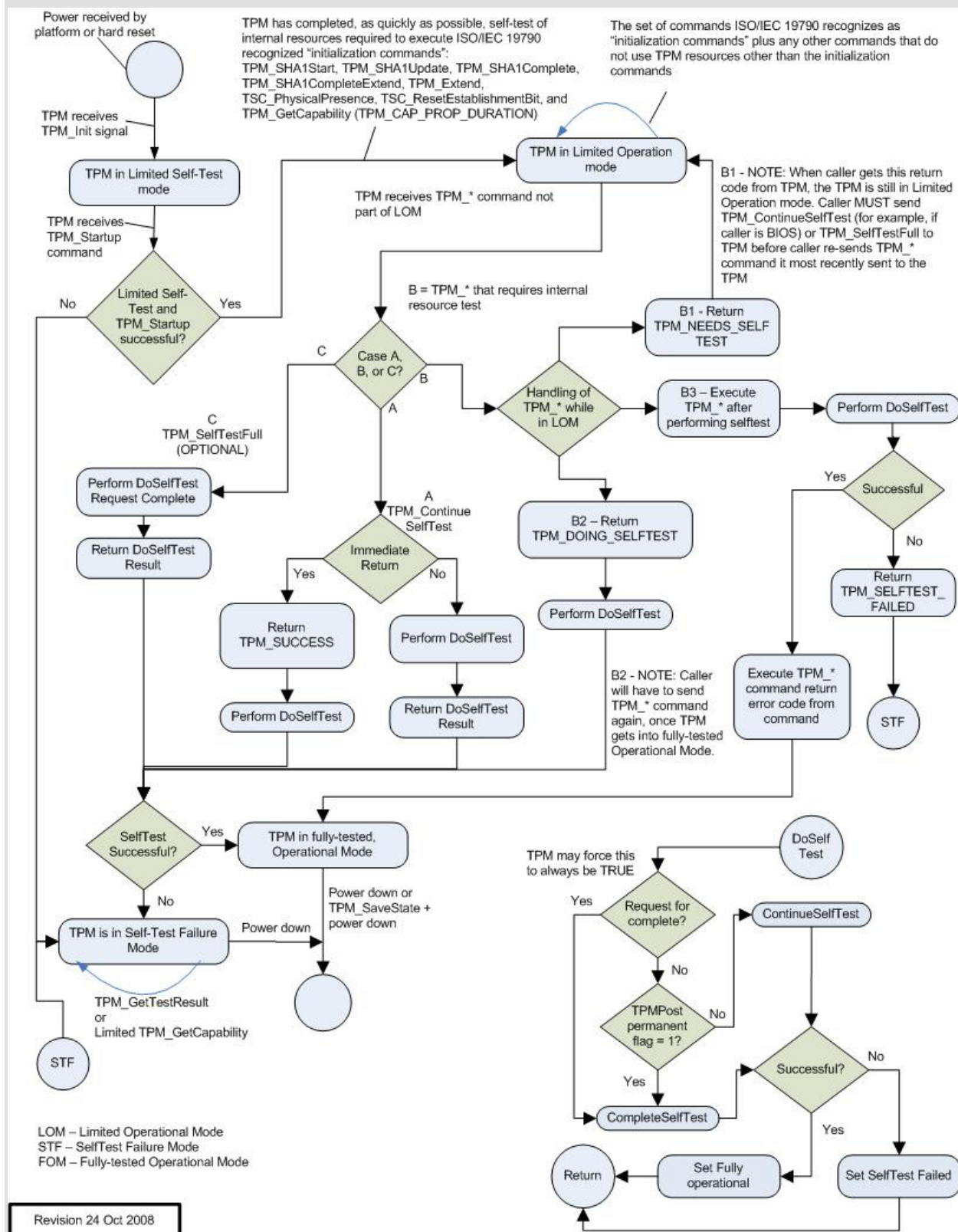


Figure 10:b - Self-Test States

After initialization the TPM performs a limited self-test. This test provides the assurance that a selected subset of TPM commands will perform properly. The limited nature of the self-test allows the TPM to be functional in as short of time as possible. The commands enabled by this self-test are:

TPM_SHA1xxx – Enabling the SHA-1 commands allows the TPM to assist the platform startup code. The startup code may execute in an extremely constrained memory environment and having the TPM resources available to perform hash functions can allow the measurement of code at an early time. While the hash is available, there are no speed requirements on the I/O bus to the TPM or on the TPM itself so use of this functionality may not meet platform startup requirements.

TPM_Extend – Enabling the extend, and by reference the PCR, allows the startup code to perform measurements. Extending could use the SHA-1 TPM commands or perform the hash using the main processor.

TPM_Startup – This command must be available as it is the transition command from the initial environment to the limited operational state.

TPM_ContinueSelfTest – This command causes the TPM to complete the self-tests on all other TPM functions. If TPM receives a command, and the self-test for that command has not been completed, the TPM may implicitly perform the actions of the TPM_ContinueSelfTest command.

TPM_SelfTestFull – A TPM MAY allow this command after initialization, but typically TPM_ContinueSelfTest would be used to avoid repeating the limited self tests.

TPM_GetCapability – A subset of capabilities can be read in the limited operation state.

The complete self-test ensures that all TPM functionality is available and functioning properly.

End of informative comment

1. At startup, a TPM MUST self-test all internal functions that are necessary to do TPM_SHA1Start, TPM_SHA1Update, TPM_SHA1Complete, TPM_SHA1CompleteExtend, TPM_Extend, TPM_Startup, TPM_ContinueSelfTest, a subset of TPM_GetCapability, and TPM_GetTestResult.
2. The TSC_PhysicalPresence and TSC_ResetEstablishmentBit commands do not operate on shielded-locations and have no requirement to be self-tested before any use.
3. The TPM MAY allow TPM_SelfTestFull to be used before completion of the actions of TPM_ContinueSelfTest.
4. The TPM MAY implicitly run the actions of TPM_ContinueSelfTest upon receipt of a command that requires untested resources.
5. The platform specific specification MUST define the maximum startup self-test time.

10.2.1 Operational Self-Test

Start of informative comment

The completion of self-test is initiated by TPM_ContinueSelfTest. The TPM MAY allow TPM_SelfTestFull to be issued instead of TPM_ContinueSelfTest.

TPM_ContinueSelfTest is the command issued during platform initialization after the platform has made use of the early commands (perhaps for an early measurement), the platform is now performing other initializations, and the TPM can be left alone to complete the self-tests. Before any command other than the limited subset is executed, all self-tests must be complete.

TPM_SelfTestFull is a request to have the TPM perform another complete self-test. This test will take some time but provides an accurate assessment of the TPM's ability to perform all operations.

The original design of TPM_ContinueSelfTest was for the TPM to test those functions that the original startup did not test. The TCG contracted with a set of FIPS-140-2 laboratories to evaluate the specification and one specific requested change was for TPM_ContinueSelfTest to perform a complete self-test. The rationale is that the original tests are only part of the initialization of the TPM; if they fail, the TPM does not complete initialization. Performing a complete test after initialization meets the FIPS-140-2 requirements. The TPM may work differently in FIPS mode or the TPM may simply write the TPM_ContinueSelfTest command such that it always performs the complete check.

TPM_ContinueSelfTest causes a test of the TPM internal functions. When TPM_ContinueSelfTest is asynchronous, the TPM immediately returns a successful result code before starting the tests. When testing is complete, the TPM does not return any result. When TPM_ContinueSelfTest is synchronous, the TPM completes the self-tests and then returns a success or failure result code.

The TPM may reject any command other than the limited subset if self test has not been completed. Alternatively, the actions of TPM_ContinueSelfTest may start automatically if the TPM receives a command and there has been no testing of the underlying functionality. If the TPM implements this implicit self-test, it may immediately return a result code indicating that it is doing self-test. Alternatively, it may do the self-test, then do the command, and return only the result code of the command.

Programmers of TPM drivers should take into account the time estimates for self-test and minimize the polling for self-test completion. While self-test is executing, the TPM may return an out-of-band “busy” signal to prevent command from being issued. Alternatively, the TPM may accept the command but delay execution until after the self-test completes. Either of those alternatives may appear as if the TPM is blocking to upper software layers. Alternatively, the TPM may return an indication that is doing a self-test.

Upon the completion of the self-tests, the result of the self-tests are held in the TPM such that a subsequent call to TPM_GetTestResult returns the self-test result.

In version 1.1, there was a separate command to create a signed self-test, TPM_CertifySelfTest. Version 1.2 deprecates the command. The new use model is to perform TPM_GetTestResult inside of a transport session and then use TPM_ReleaseTransportSigned to obtain the signature.

If self-tests fail, the TPM goes into failure state and does not allow most other operations to continue. The TPM_GetTestResult will operate in failure mode so an outside observer can obtain information as to the reason for the self-test failure.

A TPM may take three courses of action when presented with a command that requires an untested resource.

1. The TPM may return TPM_NEEDS_SELFTEST, indicating that the execution of the command requires TPM_ContinueSelfTest.
2. The TPM may implicitly execute the self-test and return a TPM_DOING_SELFTEST return code, causing the external software to retry the command.
3. The TPM may implicitly execute the self-test, execute the ordinal, and return the results of the ordinal.

The following example shows how software can detect either mechanism with a single piece of code

1. SW sends TPM_xxx command
2. SW checks return code from TPM
3. If return code is TPM_DOING_SELFTEST, SW attempts to resend
 - a. If the TIS times out waiting for TPM ready, pause for self-test time then resend
 - b. if TIS timeout, then error

4. else if return code is TPM_NEEDS_SELFTEST

a. Send TPM_ContinueSelfTest

5. else

a. Process the ordinal return code

End of informative comment

1. 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.
2. 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.
3. The TPM MUST complete the required checks before a given feature is in use. If a function self-test is not complete the TPM MUST return TPM_NEEDS_SELFTEST or TPM_DOING_SELFTEST, or do the self-test before using the feature.
4. There are two sections of startup self-tests: required and recommended. The recommended tests are not a requirement due to time constraints. The TPM manufacturer should perform as many tests as possible within the time constraints.
5. The TPM MUST report the tests that it performs.
6. The TPM MUST provide a mechanism to allow self-test to execute on request by any challenger.
7. The TPM MUST provide for testing of some operations during each execution of the operation.
8. The TPM MUST check the following:
 - a. RNG functionality
 - b. Reading and extending the integrity registers. The self-test for the integrity registers will leave the integrity registers in a known state.
 - c. Testing the EK integrity, if it exists
 - i. This requirement specifies that the TPM will verify that the endorsement key pair can encrypt and decrypt a known value. This tests the RSA engine. If the EK has not yet been generated the TPM action is manufacturer specific.
 - d. The integrity of the protected capabilities of the TPM
 - i. This means that the TPM must ensure that its “microcode” has not changed, and not that a test must be run on each function.
 - e. Any tamper-resistance markers
 - i. 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.
9. The TPM SHOULD check the following:
 - a. The hash functionality
 - i. 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.
 - ii. The TPM MAY support a test using external data.
 - b. Any symmetric algorithms
 - i. This check will use known data with a known key or a random key to encrypt and decrypt the data

- c. Any additional asymmetric algorithms
 - i. This check will use known data to encrypt and decrypt.
- d. The key-wrapping mechanism
 - i. The TPM should wrap and unwrap a key. The TPM MUST NOT use the endorsement key pair for this test.
- e. Any other internal mechanisms

10. Self-Test Failure

- a. When the TPM detects a failure during any self-test, the TPM MUST enter shutdown mode. This shutdown mode will allow only the following operations to occur:
 - i. Update. The update function MAY replace invalid microcode, providing that the parts of the TPM that provide update functionality have passed self-test.
 - ii. TPM_GetTestResult. This command can assist the TPM manufacturer in determining the cause of the self-test failure.
 - iii. TPM_GetCapability may return limited information as specified in the ordinal.
 - iv. All other operations will return the error code TPM_FAILEDSELFTEST.
 - b. The TPM MUST leave failure mode only after receipt of TPM_Init followed by TPM_Startup(ST_CLEAR).
11. Prior to the completion of the actions of TPM_ContinueSelfTest the TPM MAY respond in two ways
- a. The TPM MAY automatically invoke the actions of TPM_ContinueSelfTest.
 - i. The TPM MAY return TPM_DOING_SELFTEST.
 - ii. The TPM may complete the self-test, execute the command, and return the command result.
 - b. The TPM MAY return the error code TPM_NEEDS_SELFTEST

10.3 Startup

Start of informative comment

Startup transitions the TPM from the initialization state to an operational state. The transition includes information from the platform to inform the TPM of the platform operating state. TPM_Startup has three options: Clear, State and Deactivated.

The Clear option informs the TPM that the platform is starting in a “cleared” state or most likely a complete reboot. The TPM is to set itself to the default values and operational state specified by the TPM Owner.

The State option informs the TPM that the platform is requesting the TPM to recover a saved state and continue operation from the saved state. The platform previously made the TPM_SaveState request to the TPM such that the TPM prepares values to be recovered later.

The Deactivated state informs the TPM that it should not allow further operations and should fail all subsequent command requests. The Deactivated state can only be reset by performing another TPM_Init.

End of informative comment

10.4 Operational Mode

Start of informative comment

After the TPM completes both TPM_Startup and self-tests, the TPM is ready for operation.

There are three discrete states, enabled or disabled, active or inactive and owned or unowned. These three states when combined form eight operational modes.



Figure 10:c - Eight Modes of Operation

S1 is the fully operational state where all TPM functions are available. S8 represents a mode where all TPM features (except those to change the state) are off.

Given the eight modes of operation, the TPM can be flexible in accommodating a wide range of usage scenarios. The default delivery state for a TPM should be S8 (disabled, inactive and unowned). In S8, the only mechanism available to move the TPM to S1 is having physical access to the platform.

Two examples illustrate the possibilities of shipping combinations.

Example 1

The customer does not want the TPM to attest to any information relative to the platform. The customer does not want any remote entity to attempt to change the control options that the platform owner is setting. For this customer the platform manufacturer sets the TPM in S8 (disabled, deactivated and unowned).

To change the state of the platform the platform owner would assert physical presence and enable, activate and insert the TPM Owner shared secret. The details of how to change the various modes is in subsequent sections.

This particular sequence gives maximum control to the customer.

Example 2

A corporate customer wishes to have platforms shipped to their employees and the IT department wishes to take control of the TPM remotely. To satisfy these needs the TPM should be in S5 (enabled, active and unowned). When the platform connects to the corporate LAN the IT department would execute the TPM_TakeOwnership command remotely.

This sequence allows the IT department to accept platforms into their network without having to have physical access to each new machine.

End of informative comment

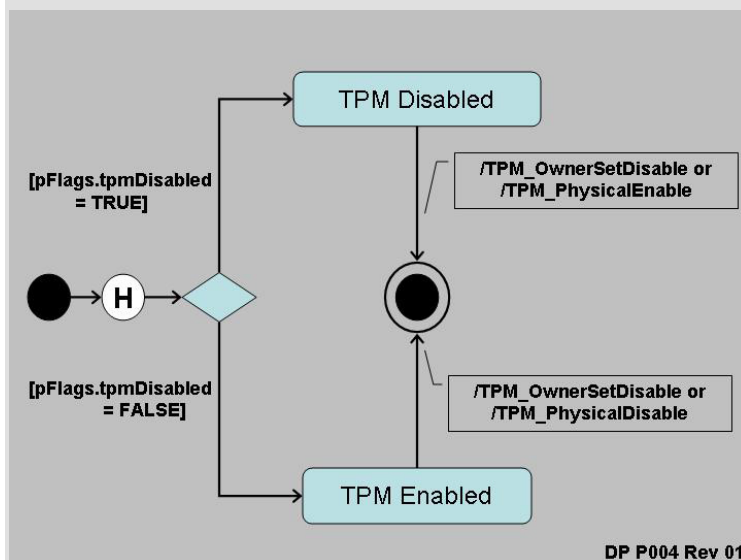
The TPM MUST have commands to perform the following:

1. Enable and disable the TPM. These commands MUST work as TPM Owner authorized or with the assertion of physical presence
2. Activate and deactivate the TPM. These commands MUST work as TPM Owner authorized or with the assertion of physical presence
3. Activate and deactivate the ability to take ownership of the TPM
4. Assert ownership of the TPM.

10.4.1 Enabling a TPM

Informative comment

A disabled TPM is not able to execute commands that use the resources of a TPM. While some commands are available (SHA-1 for example) the TPM is not able to load keys and perform TPM_Seal and other such operations. These restrictions are the same as for an inactive TPM. The difference between inactive and disabled is that a disabled TPM is unable to execute the TPM_TakeOwnership command. A disabled TPM that has a TPM Owner is not able to execute normal TPM commands.



`pFlags.tpmDisabled` contains the current enablement status. When set to TRUE the TPM is disabled, when FALSE the TPM is enabled.

Changing the setting `pFlags.tpmDisabled` has no effect on any secrets or other values held by the TPM. No keys, monotonic counters or other resources are invalidated by changing TPM enablement. There is no guarantee that session resources (like transport sessions) survive the change in enablement, but there is no loss of secrets.

The `TPM_OwnerSetDisable` command can be used to transition in either Enabled or Disabled states. The desired state is a parameter to `TPM_OwnerSetDisable`. This command requires `TPM_Owner-Authentication` to operate. It is suitable for post-boot and remote invocation.

An unowned TPM requires the execution of `TPM_PhysicalEnable` to enable the TPM and `TPM_PhysicalDisable` to disable the TPM. Operators of an owned TPM can also execute these two commands. The use of the physical commands allows a platform operator to disable the TPM without TPM Owner authorization.

TPM_PhysicalEnable transitions the TPM from Disabled to Enabled state. This command is guarded by a requirement of operator physical presence. Additionally, this command can be invoked by a physical event at the platform, whether or not the TPM has an Owner or there is a human physically present. This command is suitable for pre-boot invocation.

TPM_PhysicalDisable transitions the TPM from Enabled to Disabled state. It has the same guard and invocation properties as TPM_PhysicalEnable.

The subset of commands the TPM is able to execute is defined in the structures document in the persistent flag section.

Misuse of the disabled state can result in denial-of-service. Proper management of Owner AuthData and physical access to the platform is a critical element in ensuring availability of the system.

End of informative comment

1. The TPM MUST provide an enable and disable command that is executed with TPM Owner authorization.
2. The TPM MUST provide an enable and disable command this is executed locally using physical presence.

10.4.2 Activating a TPM

Informative comment

A deactivated TPM is not able to execute commands that use TPM resources. A major difference between deactivated and disabled is that a deactivated TPM CAN execute the TPM_TakeOwnership command.

Deactivated may be used to prevent the (obscure) attack where a TPM is readied for TPM_TakeOwnership but a remote rogue manages to take ownership of a platform just before the genuine owner, and immediately has use of the TPM's facilities. To defeat this attack, a genuine owner should set `disable==FALSE`, `ownership==TRUE`, `deactivate==TRUE`, execute TPM_takeOwnership, and then set `deactivate==FALSE` after verifying that the genuine owner is the actual TPM owner.

Activation control is with both persistent and volatile flags. The persistent flag is never directly checked by the TPM, rather it is the source of the original setting for the volatile flag. During TPM initialization the value of `pFlags.tpmDeactivated` is copied to `vFlags.tpmDeactivated`. When the TPM execution engine checks for TPM activation, it only references `vFlags.tpmDeactivated`.

Toggling the state of `pFlags.tpmDeactivated` uses TPM_PhysicalSetDeactivated. This command requires physical presence. There is no associated TPM Owner authenticated command as the TPM Owner can always execute TPM_OwnerSetDisabled which results in the same TPM operations. The toggling of this flag does not affect the current operation of the TPM but requires a reboot of the platform such that the persistent flag is again copied to the volatile flag.

The volatile flag, `vFlags.tpmDeactivated`, is set during initialization by the value of `pFlags.tpmDeactivated`. If `vFlags.tpmDeactivated` is TRUE the only way to reactivate the TPM is to reboot the platform and have pFlags reset the vFlags value.

If `vFlags.tpmDeactivated` is FALSE, running TPM_SetTempDeactivated will set `vFlags.tpmDeactivated` to TRUE and then require a reboot of the platform to reactivate the platform.

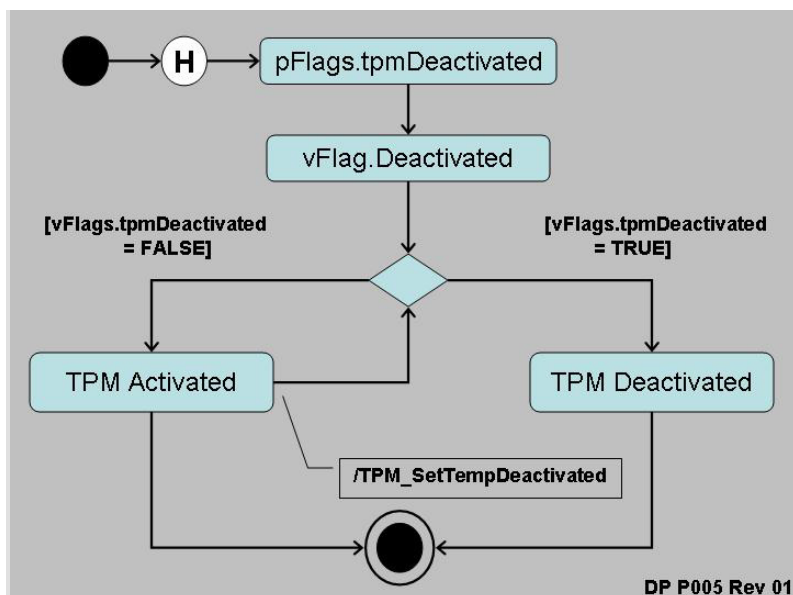


Figure 10:d - Activated and Deactivated States

TPM activation is for Operator convenience. It allows the operator to deactivate the platform (temporarily, using TPM_SetTempDeactivated) during a user session when the operator does not want to disclose platform or attestation identity. This provides operator privacy, since PCRs could provide cryptographic proof of an operation. PCRs are inaccessible when a TPM is deactivated. They cannot be used for authorization, nor can they be read. The reboot required to activate a TPM also resets the PCRs.

The subset of commands that are available when the TPM is deactivated is contained in the structures document. The TPM_TakeOwnership command is available when deactivated. The TPM_Extend command is available when deactivated so that software (e.g. a BIOS) can run the command without the need to handle an error. The PCR extend operation is irrelevant, since the resulting PCR value cannot be used.

End of informative comment

1. The TPM MUST maintain a non-volatile flag that indicates the activation state
2. The TPM MUST provide for the setting of the non-volatile flag using a command that requires physical presence
3. The TPM MUST sets a volatile flag using the current setting of the non-volatile flag.
4. The TPM MUST provide for a command that deactivates the TPM immediately
5. The only mechanism to reactivate a TPM once deactivated is to power-cycle the system.

10.4.3 Taking TPM Ownership

Start of informative comment

The owner of the TPM has ultimate control of the TPM. The owner of the TPM can enable or disable the TPM, create AIK and set policies for the TPM. The process of taking ownership must be a tightly controlled process with numerous checks and balances.

The protections around the taking of ownership include the enablement status, specific persistent flags and the assertion of physical presence.

Control of the TPM revolves around knowledge of the TPM_Owner-Authentication value. Proving knowledge of authentication value proves the calling entity is the TPM Owner. It is possible for more than one entity to know the TPM_Owner-Authentication value.

The TPM provides no mechanisms to recover a lost TPM_Owner-Authentication value.

Recovery from a lost or forgotten TPM_Owner-Authentication value involves removing the old value and installing a new one. The removal of the old value invalidates all information associated with the previous value. Insertion of a new value can occur after the removal of the old value.

A disabled and inactive TPM that has no TPM Owner cannot install an owner.

To invalidate the TPM_Owner-Authentication value use either TPM_OwnerClear or TPM_ForceClear.

End of informative comment

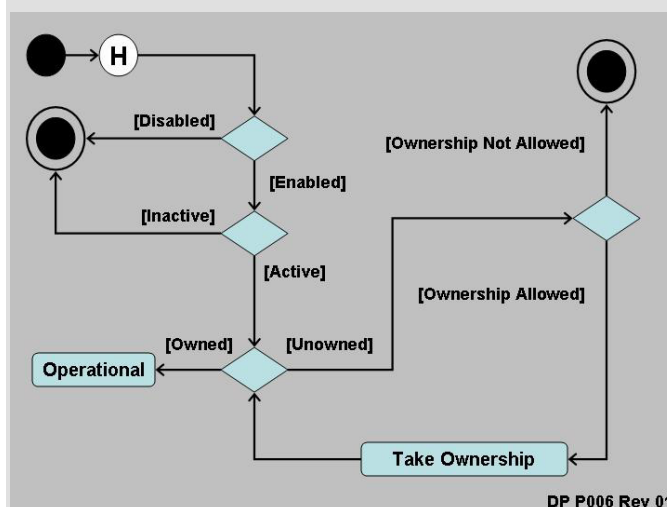
1. The TPM_Owner-Authentication value MUST be 160-bits
2. The TPM_Owner-Authentication value MUST be held in persistent storage
3. The TPM MUST have no mechanisms to recover a lost TPM_Owner-Authentication value

10.4.3.1 Enabling Ownership

Informative comment

The state that a TPM must be in to allow for TPM_TakeOwnership to succeed is; enabled and fFlags.OwnershipEnabled TRUE.

The following diagram shows the states and the operational checks the TPM makes before allowing the insertion of the TPM Ownership value.



The TPM checks the disabled flag and then the inactive flag. If the flags indicate enabled then the TPM checks for the existence of a TPM Owner. If an Owner is not present the TPM then checks the OwnershipDisabled flag. If TRUE the TPM_TakeOwnership command will execute.

While the TPM has no Owner but is enabled and active there is a limited subset of commands that will successfully execute.

The TPM_SetOwnerInstall command toggles the state of the pFlags.OwnershipDisabled. TPM_SetOwnerInstall requires the assertion of physical presence to execute.

End of informative comment