

# Trusted Computing Platform Alliance (TCPA)

## *TCPA Design Philosophies and Concepts Version 1.0*

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

All rights reserved.

### DISCLAIMERS:

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

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

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

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

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



## Table of Contents

1.	Vision .....	1
1.1	Disclaimer .....	1
1.2	Introduction .....	1
1.3	The Trusted Platform Concept.....	1
1.4	Trusted Subsystem Usefulness .....	3
1.5	Subsystem Security .....	4
1.5.1	The Common Criteria .....	4
1.6	Overall Platform Security .....	4
2.	Design Goals and Principles .....	4
2.1	Introduction .....	4
2.2	The Concept of Identity.....	5
2.2.1	Conventional and Cryptographic Identities.....	5
2.2.2	Subsystem Identities .....	6
2.3	Public Key Infrastructure.....	6
2.4	Subsystem Entities .....	6
2.4.1	The Roles .....	7
2.4.2	The Chain of Trust.....	8
2.5	TPM identities .....	9
2.5.1	Obtaining a TPM identity .....	9
2.5.2	Authorising use of a TPM identity.....	11
2.5.3	Checking a TPM identity.....	11
2.5.4	Transferring a TPM identity .....	11
2.6	Integrity .....	12
2.6.1	Authenticated and Secure Initialization/Boot.....	12
2.6.2	Recording Integrity Metrics .....	12
2.6.3	Measuring Integrity Metrics .....	13
2.6.4	Integrity Metrics in a Personal Computer .....	14
2.7	Verifying Platform Identity and Integrity.....	17
2.7.1	Obtaining integrity information from a platform .....	17
2.7.2	Verifying integrity information .....	18
2.8	Other Services .....	19
2.8.1	Associating Data with an Integrity Metric .....	19
2.8.2	Protected Storage.....	21
2.8.3	Maintenance of Capabilities .....	22
2.9	Tamper Protection .....	22
2.10	General Subsystem Capabilities.....	22
2.11	Optional Uses for the Subsystem .....	23
2.11.1	Key Usage Examples with a TPM .....	23
2.11.2	A Private-Object Store.....	23
2.11.3	Support for Security Protocols.....	24
2.11.4	Protection by an Operating System.....	24
2.11.5	Checking the State of the Operating System .....	24
2.11.6	Embedding a TPM in other security functions.....	25
2.12	Data Structures .....	26
2.12.1	Credential Descriptions .....	26
2.12.2	Data Descriptions .....	27
2.12.3	Data Initialization and Maintenance .....	28
3.	Glossary .....	29

# 1. Vision

## 1.1 Disclaimer

*This document contains no normative statements about a TCPA Subsystem:* nothing in this document is intended to imply the status of any particular data or method for compliance with the TCPA specification.

## 1.2 Introduction

This section describes the philosophies and concepts of this specification activity.

Broadly speaking, the purpose of this specification activity is to encourage the use of computer platforms for critical purposes, by improving the basis on which a computing environment may be trusted.

This specification defines a Subsystem that can be trusted to operate as expected. When correctly embedded in a platform, this Subsystem reliably measures and reports information about the environment in that platform. Information about the current platform environment is used in two ways. First, the Subsystem enhances access controls on information stored on the platform. This is done by enabling the association of stored data with a particular platform environment, such that the data cannot be accessed if the platform environment is inappropriate. Second, another entity may compare such environment information with other statements describing the platform when it is operating properly. If the reported information matches the expected information, the implication is that the platform is actually operating as expected. This is expected to increase the confidence of entities when interacting with the platform.

The Subsystem uses cryptographic processes for its internal operations, and exports the common basic cryptographic functions (except symmetric confidentiality) for use by the platform.

The Subsystem's properties are particularly useful for connected platforms.

An industry specification for a trusted platform is necessary because a sensible layperson should trust only those systems that have been publicly examined by the (cryptographic and security) community. Standardization also helps to educate customers and enables them to appreciate that these features are worthwhile.

The features provided by the first version of the specification will be sufficient to enable a basic level of trust in a platform. In particular, an implementation of the specification in a PC platform will provide a level of trust that meets the basic needs of most PC owners.

## 1.3 The Trusted Platform Concept

A Trusted Platform is a platform that can be trusted by local users and by remote entities. TCPA uses a behavioral definition of trust: *an entity can be trusted if it always behaves in the expected manner for the intended purpose*. The basis for trusting a platform is a declaration by a known authority that a platform with a given identity can be trusted to measure and report the way it is operating. That operating information can be associated with data stored on the platform, to prevent the release of that data if the platform is not operating as expected. Other authorities provide declarations that describe the operating information the platform ought to produce when it is operating properly. The local user and remote entities trust the judgment of the authorities; so, when they receive proof of the identity of the platform, information about the current platform environment, and proof about the expected platform environment, they can decide whether to trust the platform to behave in a sufficiently trustworthy and predictable manner. The local user and/or remote entities must take this decision themselves because the level of trust in a platform can vary with the intended use of that platform, and only the local user and/or remote entities know that intended purpose.

The declarations are based upon either direct or indirect assessment of a platform. A declaration may be based upon direct experience of the design and construction of a platform, or based upon other declarations.

The approach of this specification is to minimize the mechanisms in a platform that must be trusted, by the use of a separate trusted mechanism that is as small as possible so that its correct operation can be verified and trusted as much as possible. Minimization of security mechanisms is recommended security practice, because (in general) complex mechanisms are easier to subvert. The trusted mechanism uses cryptographic processes, including secrets. The design of the Subsystem deliberately avoids the use of globally shared secrets, because such secrets make it worthwhile for an attacker to break the protection afforded by a Subsystem, no matter what level of protection is provided. Application developers are encouraged to use individual secrets in their applications, but the Subsystem does not prevent applications using globally shared secrets, if their designers so choose.

The Subsystem provides protection against software attacks to the platform. To do this, the Subsystem must itself be protected against physical attack (so that it can be trusted by remote entities) and software attack (so that it can be trusted by both local and remote entities).

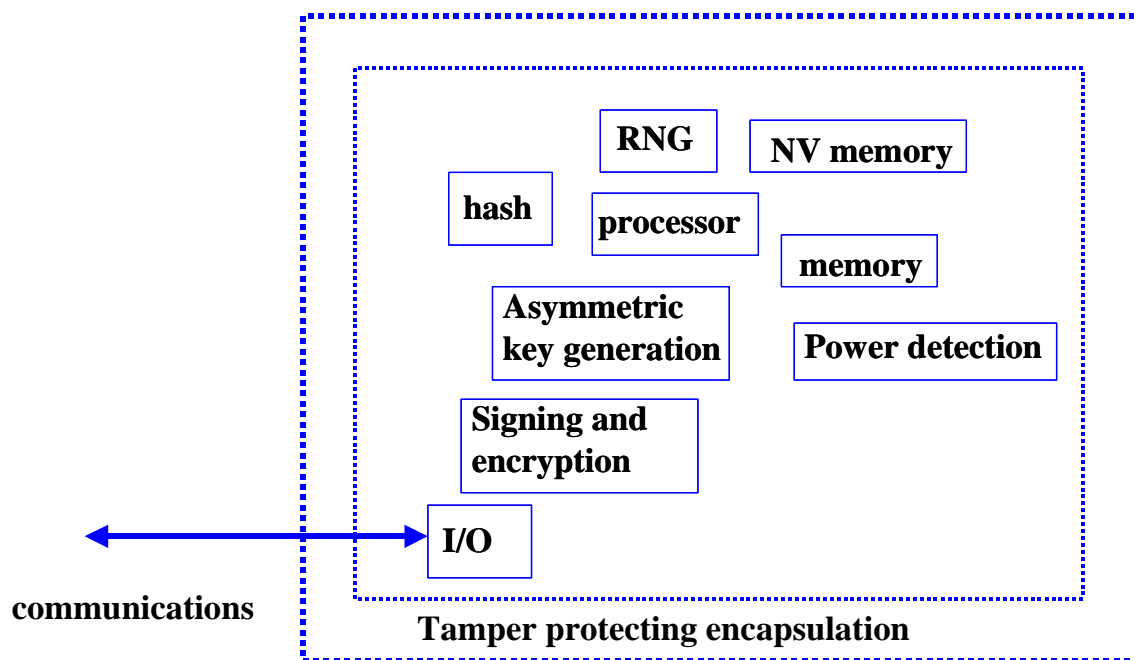
The trusted mechanisms are required to be isolated from the platform in order to protect secrets from disclosure and protect methods from subversion. The specification does not, however, prescribe any particular mixture of hardware and software methods. The highest levels of trust require additional physical resources but software is perfectly acceptable provided that the overall Subsystem has the desired properties. Generally, any software providing trusted Subsystem functionality must be loaded in a trusted state, must never damage the trusted state of the Subsystem, and must never allow secrets to be exposed to untrusted software.

The Subsystem defined in this specification must be an integral part of a platform, can be personalized by its "Owner" (the entity that controls the Subsystem, see section 2.4.1.3), and attests to its trustworthiness. Some, but not all, Subsystem capabilities must be trustworthy for the Subsystem to be trustworthy. These are called the "Trusted Set" (TS). Other capabilities must (obviously) work properly if the Subsystem is to work properly, but they do not affect the level of trust in a Subsystem. These are called the "Support Set" (Trusted platform Support Set, or TSS).

The Trusted Set of capabilities can be partitioned into measurement capabilities, reporting capabilities, and storage capabilities (and assorted cryptographic and administrative capabilities). The trusted measurement capabilities are called the "Root of Trust for Measurement" (RTM). The trusted reporting capabilities are called the "Root of Trust for Reporting" (RTR). The trusted storage capabilities are called the "Root of Trust for Storage" (RTS). The RTM makes reliable measurements about the platform and puts the measurement results into the RTR. The RTR prevents unauthorized changes to the measurement results, and reliably reports those measurement results. The RTS provides methods to minimize the amount of trusted storage that is required. The "Root of Trust for Measurement" and the "Root of Trust for Reporting" cooperate to permit an entity to believe measurements that describe the current computing environment in the platform. An entity can assess those measurement results and compare them with values that are to be expected if the platform is operating as expected. If there is sufficient match between the measurement results and the expected values, the entity can trust computations within the platform (not just within the TS) to execute as expected.

The RTR must have a cryptographic identity in order to prove to a remote entity that RTR messages come from genuine trusted capabilities, and not from bogus trusted capabilities. Ideally, this identity serves to identify the embodiment that contains all the trusted capabilities. This is not always practical, however, and sometimes the identity distinguishes just the RTR. In such cases, trust in the other trusted capabilities is indirect: a trusted authority issues a statement that the platform containing a RTR with a given identity also contains the other trusted capabilities.

This specification describes a "Trusted Platform Module" (TPM) which contains all trusted capabilities except for the RTM. The concept of a TPM is useful because it contains all the capabilities that are common to all types of trusted platform. The operation of a TPM is defined in detail in this specification, but there is no intention whatsoever to restrict the embodiment of a TPM. A TPM might or might not be a single or multiple, physical or logical, module or modules, provided that it meets the requirements of the specification. The TPM uses cryptographic techniques to reliably report its identity and the measurement results. Since this raises privacy issues, the Subsystem includes features that provide privacy controls to the Owner.



### **A typical hardware Trusted Platform Module**

This specification uses the concept of TCPA “foundations” in the platform. These are the TCPA trusted capabilities that are specific to a particular type of platform. The foundations include the method by which a TPM is incorporated into a platform, the type of the RTM, and the method by which the RTM is incorporated into the platform.

This specification defines the RTM at a very high level, only. This is because the construction and operation of the RTM is necessarily specific to a particular type of platform, and it is difficult (if not impossible) to define in abstract detail.

## **1.4 Trusted Subsystem Usefulness**

Existing software-based security services make the implicit assumption that a platform is trusted. They provide application-level security on the assumption that they execute in a safe environment. This assumption is true enough to justify the level of security required for existing business models, but state-of-the-art security functions are already providing the highest levels of protection that are possible without additional hardware support. A Subsystem in a platform provides increased confidence and enables enhancements of existing services and provision of new services. The Subsystem can support new mechanisms such as enhanced auditing and logging of software processes, platform boot integrity, file integrity, and software licensing. The Subsystem provides a protected information store for the platform and can attest to the integrity of the platform. These features encourage third parties to grant access by the platform to information that would otherwise be denied to the platform. A sophisticated Subsystem may also provide cryptographic acceleration functions.

The “integrity check” feature of the Subsystem complements software-only security services. The Subsystem contains an isolated computing engine whose processes can be trusted because they cannot be altered. These processes and the binding of the Subsystem to the platform can combine to reliably measure and report the state of the main computing environment inside the platform. The Subsystem provides a root of trust for the booting of the platform. While it is the Owner’s responsibility to provide a safe operating system for a platform, once the OS has loaded, it can report the loading of untrusted software to the Subsystem *before* that untrusted software is loaded. The Subsystem can therefore report

measured data that indicates the current state of the main software environment in the platform. A local or remote entity can simply query the Subsystem to reliably obtain these measurements and decide whether the platform's behavior enables it to be trusted for the intended purpose. Confidence in the loading of software is improved, because the Subsystem can attest to the current state of the operating system. The Subsystem also could allow the OS to boot only into a known state.

The "protected store" feature of the Subsystem also complements software-only security services. The Subsystem can act as a portal to confidential data, and will allow the release or use of that data only in the presence of a particular combination of access rights and software environment. Of course, the protected store can be used for any sensitive data, not just identity information. The Subsystem could export these services to system-level software security services (such as IPSec) that are themselves called as services by ordinary applications. This arrangement enables greater confidence in the identity of a platform, while simultaneously allowing platform anonymity if so desired. Hence, any application that invokes proof of identity can be used with greater confidence and be allowed greater power. Applications that might benefit include electronic business, electronic cash, email, free seating, networking, platform management, single-sign-on, Virtual Private Networks, Web access, and delivery of digital content (such as movies and songs).

A platform containing the Subsystem can use it to provide support for existing integrity mechanisms such as virus detection software; for example, the Subsystem might report an indication of the virus signature file that is currently in use within the platform. The existence of that virus signature file gives a high level of assurance that certain viruses are not present in the platform.

## **1.5 Subsystem Security**

The security level of the Subsystem reflects the cost of successfully tampering with the Subsystem, exposing its secrets, and subverting its processes. This tampering could be physical or logical, and so the security level depends on both the strength of physical tamper resistance and the strength of the cryptographic processes inside the Subsystem (algorithms, key length, and unpredictability of random number generator). Cost must be weighed against the value of the data to be manipulated on the platform or actions that can be initiated from the platform.

### **1.5.1 The Common Criteria**

The comparison of security systems and the level of security provided is a difficult problem made harder by international issues. To address this problem, security organizations have joined forces to create a multinational framework with a standard notational language and prescribed evaluations. Known as the "common criteria," that standard describes a "Protection Profile," which is "what the customer wants," and a "Security Target," which describes "what the vendor supplies."

The specification includes a baseline Protection Profile from which each manufacturer can create a security target and have it evaluated. One Protection Profile will describe the level of protection afforded to the TPM and RTM, which require protection against all forms of software attack and against a stated level of physical attack. Another Protection Profile will describe the level of protection afforded to the method or methods that embed the TPM and RTM into a platform.

## **1.6 Overall Platform Security**

Specification of levels of overall platform security are desirable, but overall platform security includes operating systems, applications, physical hardware, and system attributes. This system-level approach is complex and beyond the scope of the present specification.

# **2. Design Goals and Principles**

## **2.1 Introduction**

This section introduces the Subsystem in more detail.

This specification is silent on the precise embodiment of the TPM and RTM. The Protection Profiles for a platform indicate the “strength” of the methods that are used to instantiate a Subsystem and incorporate that Subsystem in a platform.

Certain critical functions in a TPM always require permission from the Owner. “Secrets” (confidential information) require authorization information before they can be used or accessed. A TPM will not function until enabled by its Owner. All secrets except for an endorsement key (described in section 2.4.1.5 “Trusted Platform Module Entity”) can be erased.

A Subsystem must be an integral part of a host platform and exports its properties to that platform. TPMs export specialized functions to the host platform and are much more immune to logical and physical interference than the rest of a typical platform. A TPM uses cryptographic operations but is not primarily a cryptographic accelerator. In fact, this specification will not require any part of a Subsystem to export (in the programming sense) a bulk confidentiality function, and will actively seek to define other functions in ways that prevent their subversion for bulk confidentiality. Like existing cryptographic modules, however, the TPM exports a variety of functions that can be trusted by both local and remote entities. This collection of properties and their derivatives are not described by existing security standards, and several aspects must be specified in this new specification.

Those new aspects are as follows:

1. The Subsystem shall enable proof of a Subsystem *identity or identities*. The Owner of a Subsystem has sole responsibility for Subsystem identities and can create new identities (and delete existing ones). An identity belonging to a Subsystem is associated exclusively with that Subsystem. Some entity vouches for the fact that a Subsystem identity belongs to a system that meets the requirements of this specification.
2. The Subsystem shall attest to the state that was used to *boot an operating system*. This enables functions to take into account the level of trustworthiness of an OS.
3. The Subsystem shall report simple integrity metrics that enable some reasoning about correct operation (or *integrity*) of the platform. Subsystems are required to reliably measure and reliably report integrity metrics. The integrity metrics available in a platform are determined exclusively by the design and construction of the platform. The actual metrics to be reported are subject to negotiation by the entity that requests the metrics. The reporting method is mandatory and defined in this specification.
4. The TPM shall have tamper protection at least at the level stated in its Protection Profile. This level of tamper protection will be reported by the Subsystem. Certain basic initialization and maintenance functions will be standardized to allow interoperability between Trusted Platforms.
5. All capabilities of Subsystems will be defined either explicitly or implicitly (by reference). This allows a customer or his proxy (such as the cryptographic or security community) to assess the level of security and make informed decisions.

The exact methods of use of the Subsystem are optional, subject to the ingenuity and needs of the market as perceived by individual manufacturers. Manufacturers can differentiate their products in the marketplace by the choice and implementation of verification method(s) and additional uses of Subsystems.

## **2.2 The Concept of Identity**

A Subsystem must have at least one externally accredited “identity” in order to take part in external authenticated communications, and may have more than one identity.

### **2.2.1 Conventional and Cryptographic Identities**

A “conventional identity” is ordinarily only a label that is unique within some context and attached to an object. In contrast, a “cryptographic identity” requires both a secret that is statistically improbable to guess and the ability to prove the possession of that secret without disclosing it. This arrangement requires an engine to operate on input data using the secret and to produce output data such that the input/output pair is statistically impossible to produce without the secret. The capability of producing data with this property is taken as proof of possession of the secret, and hence as proof of identity.



Typically, the method is a process known as digital signing, which produces a digital signature by hashing data and encrypting the digest using a secret. Examples of algorithms that can be used for signing include asymmetric algorithms (such as RSA). These can prove that data came from a specific source because different keys are used for encryption and decryption. Symmetric algorithms (such as DES) also can be used for signing, but they prove only that data came from either the source or destination of the data (because both encryption and decryption use the same key).

When asymmetric algorithms are to be used, a specific public key is the cryptographic identity. This public key is a label that is publicly known and can be used to identify a given entity. An entity proves its identity by possessing the corresponding signature key and using that private key to sign data.

### **2.2.2 Subsystem Identities**

A Subsystem may have multiple identities, each consisting of a conventional identity (a label) and a cryptographic identity (an asymmetric key). Subsystem identities are created within TPMs. At the explicit request of the Owner, some "Certification Authority" (see section 2.4.1.1) attests that each such identity belongs to a Subsystem. If a Subsystem does not have a Subsystem identity, it will not take part in authenticated communications and certain other operations. A Subsystem uses its identities according to a policy set by the Owner. Initialization of Subsystem identities is described in section 2.5.

A Subsystem cryptographic identity is required to be

- (statistically) unique,
- difficult to forge or counterfeit, and
- verifiable to either a local or remote entity.

A Subsystem achieves this by

- having capabilities that are protected from all software interference and from a limited amount of physical interference and
- using cryptographic algorithms to process data according to a secret number that is generated from a non-deterministic source.

The Subsystem uses the TPM to protect secrets and provide a cryptographic engine that operates on those secrets without disclosing them.

## **2.3 Public Key Infrastructure**

A Subsystem relies on a Public Key Infrastructure (PKI) to provide identity and integrity information in the form of credentials. The PKI is required to provide credentials that vouch for the identity of an entity and other customized credentials (defined in section 2.12.1, "Credential Descriptions") that contain more than just identity information. Credentials will be instantiated in the first version of this specification as X.509 certificates.

The provision of a PKI is outside the scope of this specification.

## **2.4 Subsystem Entities**

This section describes the entities required to support the Subsystem and provide trust in the Subsystem and its associated platform. Those entities are described in terms of roles. Roles are used to describe a Subsystem and its environment in the most general terms. For manufacturers this approach provides the opportunity for differentiation, and for customers it provides choice.

There may be a different entity for each role, or entities may have multiple roles. In some circumstances, a single entity may perform all roles. Different allocation of roles to different entities produces Subsystems (and hence platforms) with different properties. Naturally, such options affect the resources required by a Subsystem and hence its support cost.

## 2.4.1 The Roles

### 2.4.1.1 Certification Authority

A computing platform involving a Subsystem requires the support of a PKI, although a Subsystem does not itself explicitly use that PKI. This PKI provides at least one CA that vouches for the identity of other entities. *Generally, a CA enables determination of the identity of an entity by providing a certificate that binds the identity label of an entity to the cryptographic identity (public key) of that entity.*

### 2.4.1.2 Privacy CA

At least one Privacy CA is required to issue identities to Subsystems.

Any certificate that grants an identity to a Subsystem must include the statement "TCPA Subsystem identity," and the signature on the certificate must encompass that statement. The statement "TCPA Trusted Platform Module identity" is necessary to tell a third party that the CA signing the certificate vouches that the identity in the certificate belongs to a Subsystem. The choice of Privacy CA can change at any time, provided the new Privacy CA is acceptable to those parties that require evidence of identity. As usual, a chain of certificates may be required to verify the identity of another entity that uses a different CA.

### 2.4.1.3 Owner

The Owner of a Subsystem is the entity that controls the Subsystem and that may be considered its administrator. *The Owner, and only the Owner, activates the Subsystem and its identities.*

All the Owner's commands to the Subsystem must be authenticated. Owner commands are authorized by the use of a secret in an authentication protocol where the data "on the wire" is a digest of the secret and a nonce. Owner commands may be individually authenticated or submitted as part of an authenticated command session.

Ownership of the Subsystem and platform may change during the life of the equipment. For example, ownership may start with the manufacturer of the Subsystem, change to the manufacturer that installs the Subsystem into a platform, potentially change to a distributor or VAR that customizes the platform, and then change to the customer that buys the platform. When the customer sells the platform, ownership is transferred to a new Owner, and so on. An Owner takes charge of a Subsystem by erasing the previous Owner's data and reinitializing the Subsystem.

### 2.4.1.4 User

The "User" of a Subsystem is the person (or persons) who uses the platform that is associated with the Subsystem. *A User has no implicit influence over a Subsystem but instead merely uses the facilities that the Subsystem offers.* In the case of typical consumer systems, the User is generally also the Owner. In business-oriented use, the User is often different from the Owner (which, for example, may be the IT department). A User may be required to submit authorization data before the Subsystem will respond.

### 2.4.1.5 Challenger

The "Challenger" of a Subsystem is an entity that wishes to determine whether it can trust the platform for the intended purpose. *A Challenger has no influence over a Subsystem.* It merely presents an "integrity challenge" in the expectation of receiving an integrity response.

### 2.4.1.6 Validation Entity

A "Validation Entity" (VE) is an entity that vouches for part or parts of the platform associated with the Subsystem. There can be multiple VEs associated with a platform. *The VE associated with part of a platform, and only that VE, states the circumstances under which that part of the platform can be trusted for the intended purposes.* A VE does this by stating and signing the predicted values of the integrity metrics that the Subsystem should obtain from integrity measurements on a particular part of that platform.

#### 2.4.1.7 Trusted Platform Module Entity

The “Trusted Platform Module Entity” (TPME) is the entity that vouches that a TPM is actually a TPM. *The TPME, and only the TPME, provides the root of the trust in the TPM.* The TPME causes an asymmetric key pair to exist in every TPM that it wishes to endorse. The public key of that key pair is the TPM’s “public endorsement key.” The TPME signs a credential containing the public endorsement key plus the statement “TCPA Trusted Platform Module Endorsement” and supplies that credential with the TPM that it wishes to endorse. This arrangement facilitates a defense to attacks in which a public key masquerades as the identity key of a Subsystem while in fact belonging to arbitrary hardware that does not satisfy the requirements placed upon a Subsystem.

The statement “TCPA Trusted Platform Module Endorsement” is necessary to alert a third party that the enclosed key may be used to generate an identity for a TPM. The third party may wish to verify that the TPME is acceptable before taking any action as a result of receiving that data.

#### 2.4.1.8 Conformance Entity

The “Conformance Entity” (CE) is the entity that vouches that the design of a Subsystem in a platform meets the requirements of this specification. The CE assesses the overall design of the Subsystem in a product (or range of products) and issues a credential stating that the design meets this specification.

#### 2.4.1.9 Platform Entity

A “Platform Entity” (PE) attests that an individual platform was built to an accredited design.

### 2.4.2 The Chain of Trust

The root of any trust is a person or, by extension, a set of persons such as an organization or company.

#### 2.4.2.1 Certification Authority

A “Certification Authority” (CA) is the root of trust in the PKI infrastructure. Every entity uses the self-certificate of a CA in the process of verifying the cryptographic identity (public key) of another entity. The self-certificate of the CA may, of course, be just one end of a chain of certificates that ultimately proves the cryptographic identity of an entity endorsed by a different CA.

#### 2.4.2.2 Conformance Entity

The Conformance Entity (CE) is the root of trust that vouches for the design and construction of the Subsystem in a particular platform design. The CE’s integrity data is a credential that identifies the platform for conformity purposes.

#### 2.4.2.3 Validation Entities

Several Validation Entities (VEs) are the roots of trust in a platform. Each VE vouches for a particular part or parts of a platform by providing the proper value of the integrity metric that is associated with the part of the platform that is endorsed. Each VE signs a credential that includes the proper measured value and the VE’s conventional identity (a label). That “validation credential” can be stored in the platform and elsewhere. The actual location of the validation credential in a platform can vary. The validation credential from a firmware vendor (such as a BIOS vendor) would probably be part of that firmware. Alternatively, a validation credential could be stored in a preassigned place in the Subsystem or on the platform.

The Owner can persistently store in the TPM a single value that can be used to verify a list of approved values of VE integrity metrics held elsewhere in the platform. These approved values can be derived from VE certificates or generated by examining the state of a virgin platform. The Subsystem or other entity can compare the approved values with the measured values and take appropriate action if there is a discrepancy. (If the Subsystem does this, it is said to perform “secure boot,” described in more detail in section 2.6.1) Approved values can be set remotely because they are authorized by the Owner.

The Subsystem in a platform measures the integrity metrics of that platform, fetches the validation credentials from the platform, and supplies that data to any entity that challenges the integrity of the platform. Some data is protected by the TPM using cryptographic techniques.

Any Challenger can compare the measured value with the proper value from the appropriate validation credential and thus verify that the value of the integrity metric is that predicted by the VE. The policy of the Challenger may be to accept at face value any predicted value obtained from the platform. Then, no further checks are required.

Alternatively, the policy of a Challenger may be to check that the identities of the VEs are acceptable and/or to check the integrity of the validation credential. This requires the Challenger to know the identities of the VEs that it is prepared to trust and to know the cryptographic identity of each VE. The choice of trusted VEs is a matter of policy for the Challenger. Determination and validation of cryptographic identity requires the use of a CA. The choice of trusted CA is a matter of policy for the Challenger.

Note that the Challenger can be any entity, including a User or another platform.

#### **2.4.2.4 Platform Entity**

The Platform Entity (PE) vouches for the entire platform. It attests that the platform was built according to an accredited design, and indicates the precise TPM installed in the platform. The PE's integrity data includes a reference to the conformity credential supplied by the Conformance Entity and the "endorsement credential" of the TPM. This data is called the platform credential.

#### **2.4.2.5 Trusted Platform Module Entity**

The Trusted Platform Module Entity (TPME) is the root of trust when a Subsystem reports. It vouches that a particular public endorsement key may be used to send encrypted data to a genuine TPM for the purpose of loading an identity into a TPM and taking ownership of that TPM. (Data encrypted under a public endorsement key is accepted by a TPM for the purpose of loading an identity and loading the Owner's authorization data, and for no other purpose.) Using the process described as follows, this approach provides an Owner with a choice of Subsystem identities, each of which is completely isolated from the others and attests only to the fact that it is the cryptographic identity of an entity that will behave according to this specification.

As noted earlier, the public endorsement key is the public key of an asymmetric key pair that is created inside the TPM by the TPM. (The associated private key is never exposed outside the TPM and never signs or encrypts data that leaves the TPM.) While the TPM is under the control of the TPME, the TPME creates an endorsement credential by signing the public endorsement key, the statement "TCPA Trusted Platform Module Endorsement," and the TPME's identity label. The TPME supplies its endorsement credential with a TPM. Optionally, the TPME may supply its identity certificate with the TPM.

The methods embedded in a TPM permit a TPM to use the "private endorsement key" *only* for the purposes of receiving a Subsystem identity and taking Ownership of the TPM. *The TPM will not respond to requests that involve a Subsystem identity until it has been loaded with an identity.* Consequently, the Owner of a Subsystem *must* create a Subsystem identity before the remaining capabilities of the Subsystem are enabled. A Subsystem can have more than one Subsystem identity.

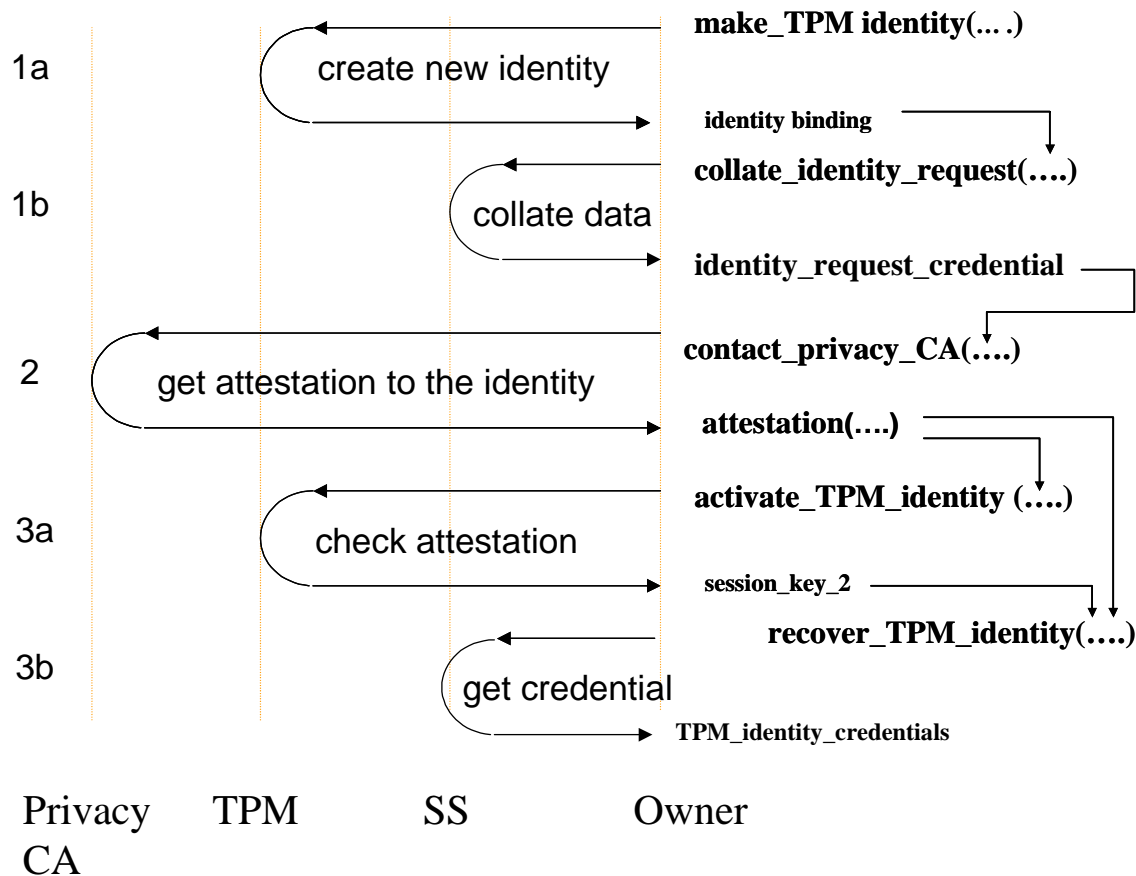
## **2.5 TPM identities**

A TPM may have multiple identities. Each identity may have attestation from exactly one Privacy CA.

### **2.5.1 Obtaining a TPM identity**

To create a Subsystem identity that is recognized by the PKI, the TPM must contain a private endorsement key. The Owner must make available the endorsement credential, the platform credential, the conformance credential, and the public key of a Privacy CA. The process of obtaining evidence of TPM identity has three main phases.

1. The Owner (a) creates the identity inside the TPM, and then (b) all necessary data is collated inside the TSS. These two stages, 1(a) and 1(b), are required for conformance to this specification.
2. The Privacy CA uses the collated data and provides credentials for the Owner. This stage (2) uses the data assembled in stages 1(a) and 1(b). This specification does not address the details of this process, but does list the actions expected of a Privacy CA. These are merely guidance, since the operation of a Privacy CA is outside the scope of a specification for a Subsystem.
3. The Owner loads (a) part of the credentials into the TPM and (b) part of the credentials into the TSS. These two stages, 3(a) and 3(b), are required for conformance to this specification.



1. The command `Make_TPM_Identity` causes the TPM to generate a new asymmetric key pair that will be the identity (public) key and the signature (private) key of a new TPM identity. It also causes the TPM to produce "identity\_binding" information that shows that an identity label is associated with a particular identity key and with a particular Privacy CA. It also loads the authorization data that must be presented in order to use the new identity. This is an atomic operation.
2. The command `Collate_Identity_Request` causes the TSS to assemble all the information that is needed for the chosen Privacy CA to make its assessment.
3. A separate protocol passes collated data to the Privacy CA, which inspects the information. If it decides to provide attestation, the Privacy CA creates the identity credentials and sends them to the Trusted Platform. The Privacy CA also creates data that will be interpreted as a statement that the credentials represent a particular TPM identity. Both the credentials and the statement

are encrypted so they can be recovered only by a TPM with a particular `private_endorsement_key`.

4. The command `Activate_TPM_Identity` causes the TPM to export a `session_key` that can be used to obtain a plaintext copy of the identity credential. `Activate_TPM_identity` exports the `session_key` only if the identity credential represents an identity of the TPM. Since the Privacy CA is trustworthy, the TPM can simply trust the Privacy CA that the supplied credential actually corresponds to a credential describing the TPM identity. This means that the TPM is not required to build or parse a certificate that attests to an identity of the TPM.
5. The command `Recover_TPM_Identity` causes the TSS to obtain a plaintext copy of the identity credential.

The important privacy features of the protocol are as follows:

1. As much control as possible is given to the Owner. The process is partitioned so that the Owner is given as many opportunities as possible to authorize the process of obtaining a TPM identity. All critical stages of the process are under the express control of the Owner.
2. The TPM cannot create an identity unless it receives explicit authorization from the Owner.
3. The secret (endorsement key) inside the TPM is used only for the process of obtaining identities (and for taking ownership of the TPM).
4. The Owner has express and explicit control over the name of TPM identities and the number of TPM identities.

The important cost features of the protocol are as follows:

- The protocol is designed to isolate those processes that must be protected capabilities. This partitioning may facilitate a reduction in cost and/or an improvement in performance.
- The data to be asymmetrically encrypted inside a TPM fits inside a single block.
- The TPM does no symmetric encryption (and hence does not require that capability).
- The TPM neither parses a certificate nor interprets ASN.1 data structures.
- The protocol does not require the TPM to remember state.
- The Privacy CA is not required to retain state information.

### **2.5.2 Authorising use of a TPM identity**

During installation of a TPM identity, the Owner installs the authorization data associated with that identity. If the Owner wishes a User to use that identity, the Owner must supply that authorization data to the User.

### **2.5.3 Checking a TPM identity**

When the Subsystem uses a Subsystem identity, the TPM uses the appropriate signature (private) key to sign data. The Subsystem may append the identity certificate to signatures so as to simplify the verification of data by third parties. Naturally, determination and validation of cryptographic identity requires the use of a CA. The choice of trusted CA is a matter of policy for the entity wishing to validate the identity of the Subsystem.

### **2.5.4 Transferring a TPM identity**

Subsystem identities cannot be transferred between TPMs except when the PE replaces a defective platform. This restriction is to prevent the copying of Subsystem identities into multiple TPMs or into entities that are not genuine Subsystems. See section 2.8.2

## 2.6 Integrity

The Subsystem measures, stores, and reports values that indicate the state of the software environment in a platform. Three data components are involved in an integrity metric. The first component is the method used to gather that data. The second component is predicted values of measured data in a platform. The third component is the actual values of measured data in a platform. Any integrity Challenger needs to know about all of these components in order to make a decision about the integrity of the platform.

Measurements must be done in ways that ensure the validity of the collected data. Hence, the Subsystem requires a root of trust for the process of measuring integrity data. This is the purpose of the RTM. The actual methods will (obviously) depend on the actual platform. In a PC, the RTM could be the BIOS boot block, the BIOS, or the TPM itself, for example. Generally, the method must prevent the subversion of the measurement process or the relevance of the measurement process. Data that is used to form an integrity metric must not include any information (such as a serial number) that might be used to identify a particular platform. As will be explained later, a log of the measurement method doesn't require special protection.

A platform must contain validation data, which is a set of the predicted values of measured data. This data depends on the properties and attributes of the components of the platform and is inserted by the Validation Entities. Each instance of validation data is wrapped in a credential and as such does not need special protection. This validation data is compared with values obtained by the Subsystem in an actual platform.

Actual values of measured data will (of course) depend on the type of platform and its configuration. The Subsystem protects such values against alteration. Hence, these values can be used for checking the consistency of the measurement log and can be compared with the predicted (approved) values.

### 2.6.1 Authenticated and Secure Initialization/Boot

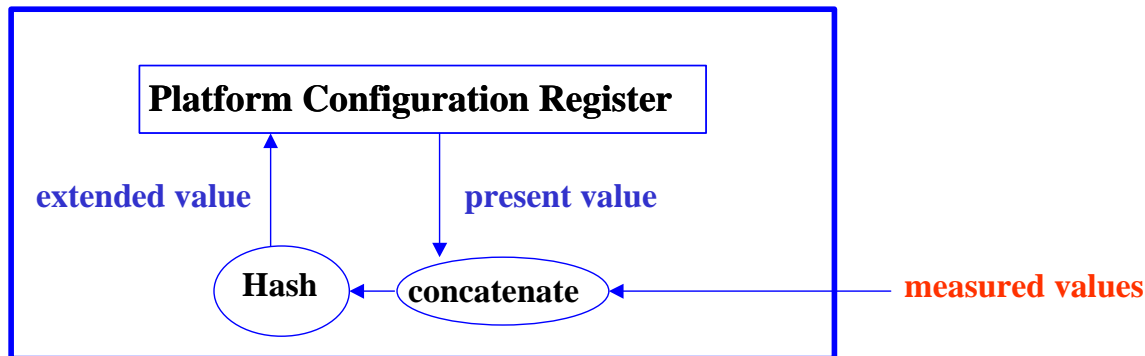
A Subsystem may use integrity metrics in either of two ways during initialization and boot:

1. In a process called "Authenticated Boot" (expected to be the norm), the trusted measurement process obtains the measured integrity metrics and the trusted reporting process reports those integrity metrics to a Challenger. The platform attempts to boot normally, irrespective of whether the measured values are the same as the predicted values.
2. In a process called "Secure Boot" (preferred in situations where it is critical that a platform always boot into a given state), the Subsystem may compare measured integrity metrics with approved values. If the measured values are the same as the approved values, the boot process continues normally. Otherwise, an exception is noted and some appropriate action is taken. The nature of that action will depend on the exact circumstances and will (obviously) require additional support procedures

An approved value may be extracted from a validation certificate, or it may be the value measured in a reference platform (after successful boot in prescribed conditions). The list of approved values is loaded into the Subsystem in some reliable manner. The Owner stores a single checksum value in the TPM. During the boot process, that checksum value is used to verify the integrity of that list, and then the list is used as the source of approved values.

### 2.6.2 Recording Integrity Metrics

## Storing values in a TPM



### Trusted Platform Module

The Subsystem provides a method to record the integrity metrics at boot time and also after an OS has loaded. In order to reduce the amount of storage necessary to hold integrity metrics and updates of integrity metrics, the TPM does not store individual integrity metrics. Instead, it stores sequences of integrity metrics. These sequences are stored in a way that tracks both the value of each integrity metric that was presented and the order in which those values were presented.

A metric is recorded in the TPM by calling an EXTEND operation. The call includes the event data that is to be incorporated into a particular sequence of integrity metrics.

In response to the call, the TPM

1. concatenates the event data to the current value of a "Platform Configuration Register" (PCR),
2. computes a digest over the concatenated value, and
3. loads that digest in the PCR.

The EXTEND operation may be called any number of times. The contents of the PCRs are reported in response to an integrity challenge. They may be used for any purpose by the Challenger.

Once an OS is running, it bears the responsibility of restricting access to the EXTEND operation to components that meet the trust requirements of the OS. This is necessary to avoid simple denial-of-service attacks that change the PCR. As an example (which is further explained in section 2.11.5), an OS Loader might treat all OS components signed by a particular authority as trusted. When such components are loaded, the OS will not call the EXTEND operation because there is no need to indicate any change in the level of trustworthiness of the OS. However, if a User demands that an unsigned driver be loaded, the OS must record that its trustworthiness has been changed to unknown and will record this fact by an appropriate EXTEND operation.

A PCR is useful because it indicates the current software state of a platform. The value in a PCR may have been caused by a change to a particular part of the platform or by rogue software in a denial-of-service attack. In either case, a third party can check the measured value against an approved value and determine whether the platform is in an acceptable state.

### 2.6.3 Measuring Integrity Metrics

A measurement agent is needed to measure integrity metrics of some part or parts of the platform and store the result in the TPM. It must also store a log in unprotected memory, which describes the measurement process. Integrity metrics can describe measurement agents, as well as parts of the platform.

A measurement agent must be a computing engine of some sort. It could be a separate microcontroller, or execute on the TPM, or could be the main platform processor itself under the influence of specific instructions, for example.



The only measurement agent that is mandatory is the RTM. This could make all integrity measurements. Alternatively, the RTM could make some measurements that encompass other measurement agents. The RTM could then pass control to those other measurement agents, which could themselves make measurements that encompass further measurement agents, and then pass control to those further measurement agents, and so on.

The RTM must be sufficiently trustworthy for the intended purpose. The strength of the RTM in a particular platform is not specified in the specification, but must be stated in the Protection Profile of a particular platform. If the RTM is the main platform processor itself, any process that executes before the RTM could completely negate the validity of integrity measurements done by the RTM or subsequent measurement agents. In such cases, it is vital to ensure that either the RTM uses the first process to execute on the platform, or that pre-RTM processes are guaranteed to be sufficiently trustworthy for the intended purpose. The trustworthiness of pre-RTM processes in such cases must be reflected in the platform's Protection Profile.

In order to stop rogue processes hiding themselves, any process that can change the level of trust in the platform must be permitted to execute only after integrity measurements on that process have been completed. If the RTM is the main platform processor itself and uses a chain of boot processes, each successive process in the boot chain could contain a measurement agent that measures some aspects of platform integrity and the integrity of the next process in the chain. After the integrity measurement of a stage in the chain has been completed (and the result stored), that stage can be executed.

Once the platform has fully booted, measurement agents must continue to measure and store integrity data of any new process that affects the level of trust in the platform, before that new process is executed.

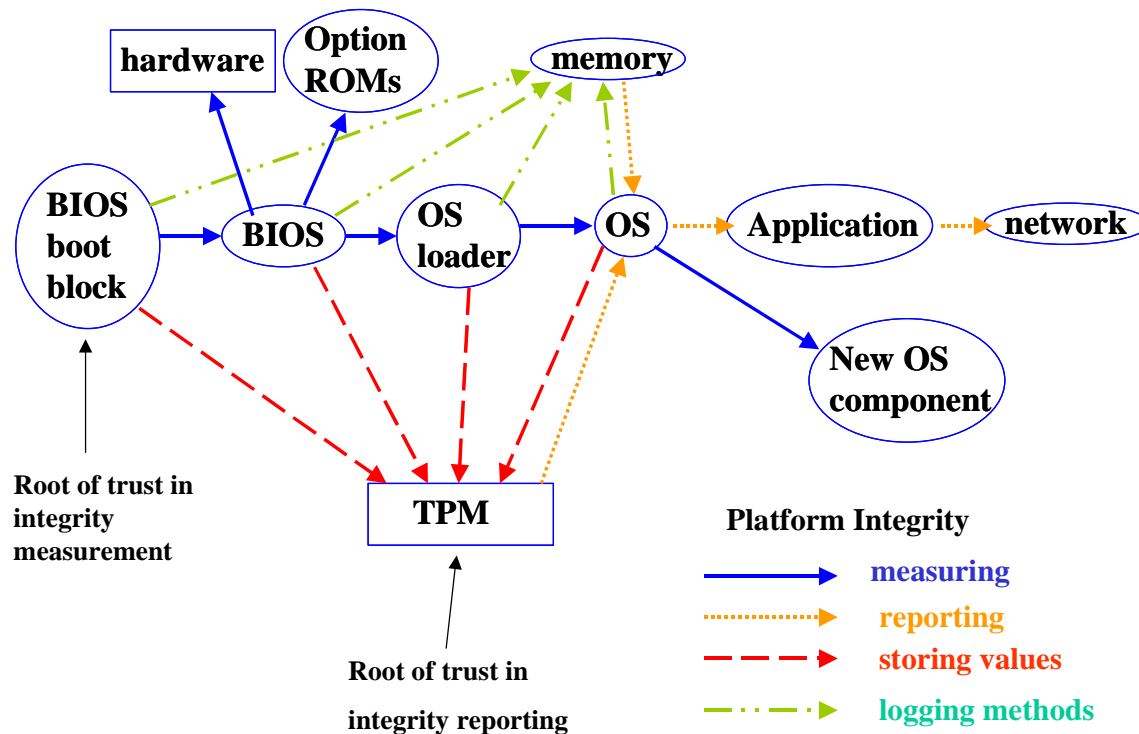
#### **2.6.4 Integrity Metrics in a Personal Computer**

The following example briefly describes one way of measuring integrity metrics in the PC architecture, in order to illustrate the meaning of previous text. The precise details of a PC specific implementation of this specification are outside the scope of this specification.

In this example the RTM is the main platform processor itself, executing instructions from the BIOS-Boot-Block (BBB). The TPM is a chip.

The Subsystem private endorsement key is inside the TPM, and an endorsement credential identifies that public endorsement key as a genuine endorsement key. Possession of the private endorsement key is therefore considered proof that the holder of the private endorsement key is a genuine TPM. The PE attests that the platform is a genuine RTM via a platform certificate that indirectly indicates: "the platform containing this specific TPM acts as a genuine RTM using the main platform processor and instructions from the BBB". This is sufficient justification for a Privacy-CA to issue an identity credential for the platform. That identity credential contains statements about the strength of the TPM, the strength of the RTM, and the strength of their incorporation into the platform. A Challenger uses the identity credential to verify that a platform contains a genuine TPM. The identity credential also convinces the Challenger that the platform also contains the RTM that is described in the identity credential.

### An example of integrity mechanisms in a (PC) Subsystem



#### 2.6.4.1 Measuring the BIOS

In this example the integrity metric is a digest of the BIOS used to boot the platform.

The purpose of the BIOS metric is to verify that the platform executed the correct BIOS immediately after execution of the BBB.

The value of the BIOS digest is used to judge whether the correct BIOS was used. This is measured by the main platform processor, acting as the RTM by executing the BBB. After performing the measurement, the BBB instructions cause the platform to use an EXTEND operation to store the digest value in the TPM. The platform also stores a record of the measurement of the BIOS in ordinary memory.

Optionally, individual blocks of functionality in the BIOS may have their own predicted digest value. Then the ensemble BIOS digest is the output of a hash process applied to the concatenation of those individual digests. This arrangement enables a policy to state the parts of the BIOS whose correct operation is critical for an intended purpose and the parts of the BIOS that are irrelevant. The order in which the individual digests are read (and the values of the individual digests) is stored in platform memory in a location known to the OS. This approach enables the OS to keep the values for inclusion in the response to an integrity challenge.

In a Secure Boot, the platform checks the computed digest against the approved value, perhaps stored as a certificate in the BIOS and verified using a public key stored in the BBB. If there is disagreement, the platform abandons the normal boot process and starts an exception process. The platform could log a Boolean value in the TPM to indicate whether the match was exact. Recording this Boolean value allows the detection of alteration of the platform from the proper state and then alteration back to the desired state.

(In other PC implementations, the RTM could be the platform executing the BIOS. This requires that the BIOS be protected against alterations by mechanisms that are sufficiently strong to justify trust in the platform for the intended purpose of the platform.)

After the BIOS has been measured, and the result stored in the TPM, the BBB passes control to the BIOS.

#### **2.6.4.2 Checking the Option ROMs**

In this example the integrity metric is a digest of all the Option ROMs used to boot the platform.

The purpose of the Option ROM integrity metric is to verify that the platform executed the correct Option ROMs immediately after execution of the BIOS.

The value of the Option ROM digest is used to judge whether the correct Option ROMs were used. The Option ROM digest is measured by the main platform processor, acting as a measurement agent by executing the BIOS. After performing the measurement, the BIOS instructions cause the platform to use an EXTEND operation to store the digest value in the TPM. The platform also stores a record of the measurement of the Option ROMs in ordinary memory.

The integrity metric in this example is an ensemble digest, calculated over individual digests of the Option ROMs. The record of the measurement of the Option ROMs includes a list of the order in which the Option ROM digests were incorporated into the ensemble digest and includes the individual digests of the Option ROMs. Each entry in the list contains fields that serve to identify the Option ROM to some degree. These fields include identification of the slot occupied by the device, the manufacturer, the type of device, and the version number, for example. The record of the measurement of the Option ROMs is logged by the platform in platform memory, in a location known to the Operating System. This enables the OS to keep the values for inclusion in the response to an integrity challenge.

A Secure Boot process could trigger an exception only if certain Option ROMs fail their integrity tests. (Option ROMs may safely be ignored if they are non-critical for the intended purpose of the platform.)

Manufacturers of Option ROMs are encouraged to include generic identification data and validation data in new products.

Existing devices may contain no useful generic identification information, and the only data to be reported may be the number of the slot occupied by the device. Identification data of future devices may or may not be stored as part of the Option ROM itself.

Validation data may be obtained from the manufacturer of the Option ROM, and may or may not be stored as part of the Option ROM itself. Alternatively, the predicted value of an individual digest may be the value measured in a reference platform, after successful boot in prescribed conditions. This may be the only choice if there is insufficient data to identify a manufacturer, or if the manufacturer chooses not to publish the predicted value. The predicted value of the ensemble digest may be the value measured in a reference platform, after successful boot in prescribed conditions. This may be the simplest choice if an organization is prepared to vouch for the actual hardware configuration of a platform. Any actual predicted value used in any comparison is the responsibility of each entity that verifies the integrity of a platform.

#### **2.6.4.3 Checking Hardware Configuration**

In this example the integrity metrics are a motherboard digest and a hardware-device digest.

The purpose of the hardware configuration integrity metrics is to verify that the platform contains the correct hardware.

The value of the hardware configuration digests is used to judge whether the platform contains the correct hardware. The hardware configuration digests are measured by the main platform processor, acting as a measurement agent by executing the BIOS. After performing the measurement, the BIOS instructions cause the platform to use an EXTEND operation to store the digest value in the TPM. The platform also stores a record of the measurement of the hardware configuration in ordinary memory.

The integrity metrics in this example are ensemble digests, calculated over individual digests of the hardware. The record of the measurement of the hardware configuration includes a list of the order in which data from the motherboard-components was incorporated into the motherboard digest, and the order in which device data was incorporated into the hardware-device digest. Each entry in the list contains fields that serve to identify the hardware component to some degree. These fields include identification of the manufacturer, the type of device, and the version number, for example. The record of the measurement of the hardware configuration is logged by the platform in platform memory, in a location known to the Operating System. This enables the OS to keep the values for inclusion in the response to an integrity challenge.

A Secure Boot process could trigger an exception if only certain devices fail their integrity tests. (Devices may safely be ignored if they are non-critical for the intended purpose of the platform.) Manufacturers of devices are encouraged to include identification data and validation data in new products.

Many existing devices or components may supply no useful identification information. Identification data of future devices may or may not be stored as part of the device itself. The motherboard manufacturer may provide validation data for the motherboard and a list of the order in which data from the motherboard-components was incorporated into the motherboard digest. That data may or may not be stored on the motherboard itself.

A device manufacturer may provide validation data for that device. That data may or may not be stored on the device itself. Alternatively, validation data may be the value measured in a reference platform, after successful boot in prescribed conditions. This may be the only choice if the manufacturer chooses not to publish the predicted value. The validation data of the ensemble digest may be the value measured in a reference platform, after successful boot in prescribed conditions. This may be the simplest choice if an organization is prepared to vouch for the actual hardware configuration of a platform. Any actual validation data used in any comparison is the responsibility of each entity that verifies the integrity of a platform.

#### **2.6.4.4 Checking the loading of the Operating System**

The actual process of loading the OS is outside the scope of this specification. Generally, however, the main platform processor acts as a measurement agent by executing the BIOS, which measures the OS Loader by computing its digest, storing the integrity data in the TPM, and then passing control to the OS Loader.

Typically, the process of loading an OS involves several components, each loading a more complex component, until the final OS is loaded. In that case, the platform acts as successive measurement agents by executing each successive component of the OS Loader, which measures and stores integrity data relating to the next component of the OS Loader.

## **2.7 Verifying Platform Identity and Integrity**

### **2.7.1 Obtaining integrity information from a platform**

An integrity challenge is used to obtain identity and integrity information from a target platform.

An integrity challenge is a typical challenge/response process. The challenge transports data that queries the identity and integrity of a platform. The response transports evidence of the identity and integrity of a platform.

There are no restrictions on the issuing of an integrity challenge. For example, an integrity challenge may be issued by a standalone program or integrated into existing applications. An integrity challenge may be issued at any time. The challenge may be at the request of a User or at the request of an application. The purpose of the challenge may be to verify the integrity of a platform or to generate a reference set of integrity metrics that will be used for future comparisons.

An integrity challenge must be sent to a particular Subsystem identity. In order for that identity to provide a challenge response, the integrity challenge must be accompanied by the authorization data needed to use the private key of that Subsystem identity. This enables an Owner to create a policy governing

responses to challenges, by controlling access to that authorization data. The Owner may specify that integrity responses can be sent to certain Challengers only, by providing the authorization data only when the challenge has come from a named Challenger, for example.

A challenging entity that wishes to verify the identity of a Subsystem and trustworthiness of the associated platform sends a token (containing at least a nonce) to the platform. The nonce will typically be a random number. The token may accompany a request for specific data whose validity depends on the trustworthiness of the associated platform

The platform receives the token and obtains the requested data (if such a request accompanied the token). The platform (perhaps via the Support Services) performs a QUOTE operation on the TPM. That causes the TPM to (in effect) append the measured integrity data and the requested data to the token, sign the combination with a signature (private) key belonging to a Subsystem identity, and return the result to the caller. The platform (perhaps via the Support Services) appends the method integrity data, optionally appends other data that simplifies the interpretation of the integrity data, and sends the package back to the Challenger.

### 2.7.2 Verifying integrity information

The processing of the integrity data by the Challenger will depend on the TPM and the challenge. The Challenger has several options:

- The policy of the Challenger may be to accept at face value all validation data in the validation certificates embedded in the platform. Such a policy takes the view that “if the predicted values of integrity metric are embedded in the platform, they’re probably correct.” In this case, all that is required is to compare the measured integrity metrics with the predicted values inside the validation certificates and report whether there is an exact match.
- The policy of the Challenger may be that only certain Validation Entities can be trusted. In this case, the identities of the VEs must be compared with a list of acceptable VEs.
- The policy of the Challenger may be that the integrity of the validation certificates must be verified. In this case, certificates containing the public keys of the VEs must be obtained from trusted CAs.
- The policy of the Challenger may be that it wishes to do some or all of the checking itself. In this case, the necessary data must be returned to the Challenger. In some Subsystems, it may be possible to allow the TPM to do comparisons and simply report the result back to the Challenger. It must be noted, however, that this specification does *not* require that the TPM be able to interpret certificates. A Challenger must take this into account when requesting the TPM to act as its proxy and may choose to supply the TPM with the approved data that is necessary for comparisons.

A step-by-step description of the processes to perform a check on an integrity response is:

1. The Challenger obtains a copy of the CA’s public key and uses it to verify the Subsystem’s identity credential. The Challenger extracts the Subsystem-identity key from the Subsystem-identity credential and uses it to verify the signature on the reflected token and integrity data. If all verifications are correct and the reflected token is the same as that in the challenge, there is acceptable evidence that the data was sent in reply to the challenge (so the Subsystem is online and there is no replay attack).
2. The integrity data from the target must be inspected to determine if the platform can be trusted for the intended purpose. The following diagram illustrates three processes.
  - One process is to reproduce the values that should be reported by the TPM, by using the primitive measurements in the measurement log in the order stated in the measurement log. The reproduced values must be compared with the signed values reported by the TPM. This checks that the measurement log is a reliable indication of the history of the target platform.

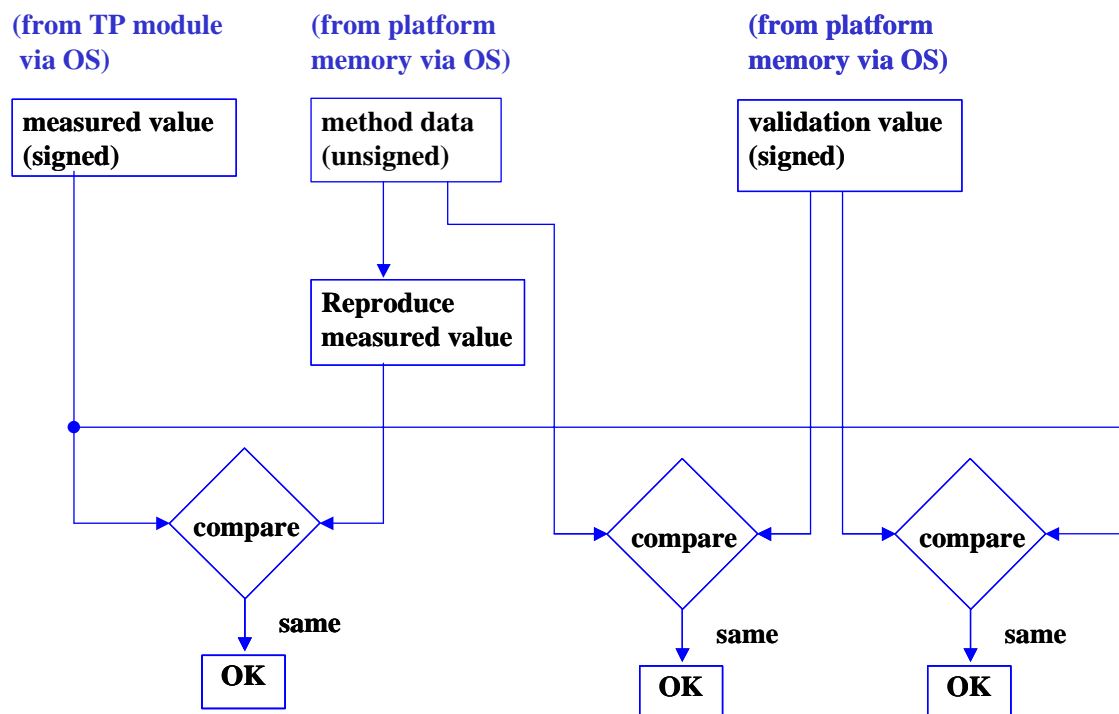
- A second process is to compare the primitive measurements in the measurement log with values stated by validation entities. This checks that the individual platform elements that were examined in the target platform are operating as predicted by the validation entities.

These two processes together are sufficient to determine the history of a target platform, and hence help decide whether the target platform is acceptable for the intended purpose.

- A third process directly compares the values reported by the TPM with values stated by validation entities. This immediately checks that TPM values are the same as those predicted by the validation entities.

This third process is a shortcut method to determining whether a target platform is acceptable for the intended purpose, and relies upon validation entities that have already determined that certain TPM values correspond to an acceptable platform history.

### Verifying an integrity value



The Subsystem Identity Credential also contains the name of the TPM's manufacturer and the type of the TPM. The Challenger may use this information in conjunction with policies to determine whether the platform may be trusted for the purpose intended by the Challenger. Such a decision is not part of this specification and is the choice of the Challenger.

## 2.8 Other Services

### 2.8.1 Associating Data with an Integrity Metric

The TPM provides a means by which data is bound to a Subsystem and a particular value of measured integrity metric. The method may be used for any purpose.

- The method may be used to store encryption keys associated with certain files, for example. This would prevent access to encrypted data on an HDD if an HDD was to be transferred to another platform, if the platform configuration was to change, or if a new software stack was to be executed.
- The method may be used to store authentication information. Software would be granted access to the authentication information only if the values of integrity metrics indicate that the correct OS had booted, for example. This would improve authentication of an OS to a remote third party.

The TPM will securely store and associate data with a specified value of integrity metric. The TPM will not release that data unless the current value of that measured integrity metric matches the specified value of that integrity metric.

This specification provides mechanisms to cope with upgrades and migration from one platform to another. *This specification does not include a mechanism for restoration from failure of a TPM, except with the cooperation of both the Owner and the PE.* Any application that uses this method should therefore provide its own means of dealing with TPM failure.

If an operating system seeks to create data that is bound to an integrity metric, the following occurs:

1. The OS issues a SEAL call to the TPM. This passes authorization data and data-to-be-bound to the TPM, specifies the target integrity metric, and specifies a value of that integrity metric. The call may specify that the value of integrity metric is the same as the current value of the target integrity metric, it may specify a different integrity metric (presumably one known to meet similar standards of trust), or it may specify *don't care*.
2. The TPM notes the current value of the specified integrity metric as a record of the current state of the platform. The TPM creates a set of credentials by associating the current state with the SEAL parameters. The TPM stores the set of credentials such that they are accessible only to the TPM.

If an OS wishes to access bound data, the following occurs:

1. The OS issues an UNSEAL request to the TPM. This passes authorization data to the TPM.
2. The TPM uses the authorization to access the credentials. The TPM does not return credentials unless the credentials are valid.
3. The TPM checks that the current value of the target integrity metric is the same as the value of the target integrity metric in the credentials.
4. If the identity metric is the one specified during the SEAL operation, the bound secret and the identity metric of the SEALing platform is revealed to the caller. Otherwise, an error is returned.

This mechanism allows access to bound data to be correctly maintained when part of the platform is upgraded and, hence, the value of an integrity metric is permanently altered. In that case, before the platform is upgraded the OS must first use an UNSEAL call to retrieve the data and then issue a new SEAL call that includes the new value of the integrity metric. Only then is the platform upgraded.

The TPM provides the highest level of security (data can be bound to a specific platform in a specific configuration), but this security is weakened if software processes in the platform, outside the TPM, can influence the binding. As a result, the migration of SEALED data to other platforms, and the backup of SEALED data in the event of catastrophic platform failure, is managed by higher-level protocols in cooperation with a third party.

For example, a network administrator might provide client operating systems with a service that accepts the (plaintext but encrypted for transport) SEALED secrets. In the event of migration or catastrophic failure, the network administrator is required to re-enable access to a client's bound data. In the case of premium data SEALED to a particular software stack and platform, the data license might specify a particular trusted authority that is charged with accepting (plaintext, but encrypted for transport) SEALED secrets. The trusted authority must carefully restrict to whom these secrets are revealed. Typically the User must present another credential to the trusted authority before the authority agrees to re-enable access to a client's bound data.

### 2.8.2 Protected Storage

Any data can be rendered confidential through encryption and protection of the key used for encryption. Similarly, any signing authority can be protected if the signing key is protected. A service that protects keys is therefore useful, and sometimes essential. Similarly, it is useful, and sometimes essential, to provide a service that protects authorization data (for example).

The Subsystem enables such a service because a TPM can act as a portal to keep arbitrary amounts of small pieces of data in a confidential format. "Protected storage" is a set of commands provided by the TPM to enable virtual secure storage space. The provision and use (and management) of a service that uses the protected-storage commands is outside the scope of this specification.

The Subsystem is required to offer protected storage as a service to functions outside the TPM. This enables applications to provide functions such as User association, key archive, and key restoration, and enables the efficient migration of Subsystem information from one platform to another within a heterogeneous PC environment.

The protected-storage service also enables a Subsystem to reduce its own requirement for storage of Subsystem keys within the TPM. Of course, a TPM might have so much internal storage that it does not need to use protected storage for this purpose. It should be noted, however, that such a choice might increase the cost of the TPM.

The only method by which functions outside the TPM can retrieve the plaintext contents of protected storage is to use the TPM as a portal.

Any off-Subsystem data will be protected with cryptographic techniques, and the process will necessarily involve cryptographic keys. The following criteria were important factors in the selection of the protected-storage method:

- Keys used to provide protected storage and the associated encryption algorithm should be arbitrarily strong to protect off-Subsystem data, and *can* be arbitrarily strong if the off-Subsystem method prevents subversion for arbitrary confidentiality.
- Keys loaded into the TPM can be backed up by the source of that data, using any conventional means (not specified here), before the data is loaded into the TPM. Hence, the TPM does not really need a backup and restoration method for external data/keys. Nevertheless, it is an advantage if Users can back up and restore data/keys that are stored as protected storage.
- The Subsystem requires a mechanism to enable reproduction of existing TPM keys in protected storage if a TPM fails during its warranty period and is replaced by an identical TPM.
- A given Subsystem identity must be guaranteed to be available to precisely one genuine TPM. So Subsystem (cryptographic) signature keys belonging to a TPM identity must not be backed up by the Owner so as to prevent an Owner from cloning Subsystem identities at will. Nor can such a Subsystem signature key be exposed to a third party, because that would potentially introduce uncertainty in the future over the validity of a signature.

In order to satisfy the objectives, this specification partitions protected data according to whether the data/keys are loaded into the TPM ("migratable data/keys") or are generated inside the TPM ("non-migratable keys").

Conceptually, backup and restoration for data/keys in protected storage involves the cooperation of a third party. A secret from the TPM is encrypted under the public key of that third party and exported from the TPM. If the third party cooperates with the Owner of an old TPM and the Owner of a new TPM, a new TPM can be "introduced" to protected-storage data created by another TPM and caused to accept that existing data.

This specification demands two such third parties. One is used to back up and restore just the migratable data/keys (loaded *into* the TPM) and is the Owner or an agent of the Owner. The other is used to back up the non-migratable data/keys (created *by* the TPM) and is the PE or an agent of the PE. This split is required because the Owner cannot be given sole responsibility of restoration of data that is supposed to be unique. Two methods of backup and restoration are defined. The method for migratable data permits



the Owner to activate the backup process at any time. The method for non-migratable data requires that the backup process be initialized by the PE and gives the Owner the right to disable that backup process.

### 2.8.3 Maintenance of Capabilities

It may be necessary from time to time to modify the capabilities installed in the TPM. Changes to TPM capabilities require authorization by both the TPME and the Owner. The TPME must authorize changes because the TPME is the entity that has endorsed the TPM. The Owner must authorize changes because the TPM belongs to the Owner.

Any alteration to a TPM capability will therefore be accepted only if it is signed first by the TPME signature (private) key and then authorized by the Owner using the TPM Owner's authorization data. Any alteration is verified by the TPM as coming from the Owner and then verified as being signed by the TPME.

Hence, the TPME must load the TPME's identity key into the TPM during manufacture of the TPM.

## 2.9 Tamper Protection

Trust in the Subsystem depends critically upon correct operation of the Trusted Services. The secrets and methods of the TS must be protected against both physical and logical attack. The level of tamper protection employed by the TS in a particular platform is described in the Protection Profiles of that platform.

The cost of hardware is expected to be proportional to the degree of tamper protection of that hardware. There is likely to be a range of hardware TPMs, therefore, with different degrees of tamper protection. A sense of proportion is required to strike the balance between the value of the information being protected and the cost of the Subsystem to provide a necessary degree of tamper resistance. It should be appreciated that it is impossible to make a Subsystem totally tamper-proof. What one person can make, another can break, given enough time and money.

Any consideration of tamper protection includes the following attributes:

- Erasing secrets in the absence of external power.
- Providing tamper resistance by techniques such as encapsulation, passivation, or routing of signals.
- Providing defenses to analytical attacks such as power analysis or timing analysis.
- Providing tamper detection by detecting attacks such as physical removal of casing, damage to physical integrity of casing, application of excess temperature (both high and low), application of out-of-specification voltages or application of X-rays.

If a TPM supports tamper detection, it must be able to report whether it has been tampered with.

## 2.10 General Subsystem Capabilities

This specification defines capabilities to provide secure storage and support identity and integrity using a Subsystem, and some capabilities to control and manage that Subsystem. The actual uses of a Subsystem are the choice of the manufacturer and are outside the scope of this specification.

A basic Subsystem

- will export a protected-storage service to other applications and services,
- will export identity functions and integrity metrics to other applications and services, but
- will not require the export of a bulk confidentiality (symmetric encryption) service.

A bulk confidentiality service may be provided at the option of the manufacturer. More sophisticated Subsystems can act as full-blown cryptographic accelerators.

There may be several integrity functions. These will be customized to the host platform hardware and the host platform software. The Subsystem will interact with the platform hardware and software in a way that enables the Subsystem to report reliably on the operation of certain parts of the platform. This enables

the Subsystem to support improved verification processes. Preferably, some of these provide access to inviolate controls and indicators that cannot be subverted by the platform (for sole use by the Subsystem).

Subsystems will also provide the means (via authorization data) to set and enforce policies controlling the operation of that Subsystem. The Owner of a Subsystem sets policies that determine what the Subsystem may or may not do. The User of a Subsystem sets policies that determine what the Subsystem may or may not do with information related to the User. These policies must be reconciled, so a User might negotiate an acceptable policy before agreeing to use a platform.

For example, the Owner might specify a range of policies, and the User chooses one that is compatible with the User's requirements. The Owner might offer a policy where the Subsystem never asks permission to disclose the User's ID, one where the Subsystem asks for permission only at the start of a session, and one where the Subsystem always asks whenever the ID is used. If the Subsystem has more than one identity, a policy will be required to determine which identity should be used in any particular circumstance.

## **2.11 Optional Uses for the Subsystem**

Not all cryptographic operations need be done inside the protected environment of a TPM. Generally, if data will eventually appear as plaintext (clear) outside the TPM and the secrets (keys) that protect that data do not protect other data, it is safe to do cryptographic processing of that data outside the TPM. All other cryptographic processes should be done inside the TPM.

For example, if a session key has been negotiated for the encryption/decryption of data that will be available to the platform as plain text, it may be acceptable to use that key outside the TPM. This approach may improve performance if the overall effect is that more processing can be done on the platform CPU. On the other hand, it is unsafe to use long-term secrets (such as identity keys) outside the TPM. Processing involving private key operations *must* be done inside the TPM. This is because those keys prove identity and must be difficult to forge or counterfeit.

### **2.11.1 Key Usage Examples with a TPM**

Here are some examples of key usage with a TPM:

- Signing with an Asymmetric Key. First load the key if it is not already in the TPM, and then sign with it.
- Generating and Storing a Key. First load the parent key in the protected storage hierarchy. Generate a new key and encrypt the new key with the parent key.
- Migrating a Key from one Platform to Another. The key to migrate is encrypted by a public key chosen by the owner.

Maintenance of TPM. To recover from damaged TPM, the TPM Owner in collaboration with the TPM Manufacturer can create an encrypted maintenance package. Installation of the package in a replacement TPM requires collaboration with the TPM Owner and TPM Manufacturer.

### **2.11.2 A Private-Object Store**

Any private object can be stored using a Subsystem. This does not imply that arbitrary private objects should be stored actually in a TPM. TPMs are required to provide protected storage, whereby the TPM acts a portal for the storage of keys and authorization information. While arbitrary private objects could be split into small pieces and stored in a TPM, masquerading as authorization information, such a method would be very inefficient compared to storing those objects in the main platform as encrypted data and using the TPM just to store keys.

Access to protected storage requires the use of authorization data. Therefore, access to private objects can have an access-control mechanism, to protect those objects from unauthorized intrusion. Although the Owner does not have access to data/keys in protected storage that were not loaded by the Owner, the Owner can always prevent the loading of such data and/or erase such data without permission from the entity that loaded the data.

### 2.11.3 Support for Security Protocols

An application can use protected storage to store secrets used by other cryptographic protocols such as IPSec and SSL.

Identity information (keys and authorization information) can be directly stored via the TPM, using protected storage. An identity store is useful because some desirable applications require the use of an identity other than that of a User, must operate when a User is absent and hence cannot personally vouch for the platform, or require concurrent identities. Some of these applications are improvements to existing processes (such as remote management) or solutions to existing problems (such as software licensing and distribution). Other applications are speculative, intended to support newer styles of business (such as free seating, multiple users and e-business).

Identity information can include other protected data such as User identifier, machine identifier and other User- and/or machine-related information.

There may be several identity functions, each matched to the peculiarities of existing security services such as email, IPSec, VPN, PPTP and so on. Each identity function, however, can be based on the storage of identity information within the Subsystem, and can operate on that identity information without exposing it outside the TPM, or expose that identity information only when the platform's software environment is in a safe state.

### 2.11.4 Protection by an Operating System

It is impossible for an operating system to guarantee that it can take appropriate action if a platform did not boot correctly, because the OS itself might have been subverted. It is possible, however, for the OS to *attempt* to take appropriate action, on the understanding that the action is a "best effort." It is also possible for the OS to take actions before subversion so as to protect data after subversion.

The best that an operating system can do is issue an integrity challenge to the Subsystem in its platform and interpret the results. If the measured metrics do not match the predicted metrics, the OS may boot a system that provides restricted functionality, in an attempt to protect platform resources. The choice of functionality will depend on the platform and the policy embedded in the OS. The functionality may include the issuing of an alert to the User and to a remote entity.

Note however, that an OS can store credentials or secrets that are accessible *only* when the platform is in a known or approved state. Accessibility is enforced by the TPM, so the OS vendor can intentionally limit software functionality if the platform boots into an unknown configuration, even if the OS has subsequently been subverted.

The method is as follows:

1. The platform QUOTE operation allows the trustworthiness of the platform to be established when the platform is online.
2. Once trust has been established, the SEAL operation can be used to set mechanisms in the TPM that can always be trusted, even when the trustworthiness of the platform cannot be established.
3. The TPM and UNSEAL functions maintain trustworthiness even when the platform is offline or has been rebooted, for example.

### 2.11.5 Checking the State of the Operating System

An operating system is typically complex and, consequently, this makes it difficult to devise meaningful integrity metrics.

While it is outside the scope of this specification, the recommendation of this specification is to use the EXTEND command to store additional integrity metrics in the OS Loader PCR whenever the OS determines that an action is about to decrease the level of trust in the OS. At the very least, this change serves to warn an integrity Challenger that the platform is no longer in the basic trusted state that existed immediately after boot of the platform. At best, the current value of the OS Loader PCR indicates to a Challenger precisely what change(s) to the platform have taken place.

The process requires the OS to distinguish between trusted and untrusted OS components, including device drivers. Trusted components may be identified if they are signed and accompanied by a certificate, and a Trusted Component Policy is available. The OS verifies that a component is trusted by verifying the component using the signature, verifying the signature using the associated certificate, and verifying that the associated certificate is signed by an entity that is listed in the Trusted Component Policy.

A Trusted Component Policy provides a list of acceptable component sources and a list of unacceptable component sources. The policy requires a default action for component sources that are in neither list. It is expected that the policy *per se* contains the addresses of Certification Authorities that provide lists, rather containing actual lists.

When the OS is loading, a digest of the policy is submitted as the argument to an EXTEND operation on the OS Loader integrity metric. This provides a method of recording the policy that was used to load the OS. After the platform has booted, and before any untrusted OS component is loaded into memory, the OS has a level of trust that depends on the design of the OS and any trusted components that have been loaded.

When a new component is about to be loaded into memory, the OS kernel checks whether it is a trusted component and then does the following:

1. If it is a trusted component, the OS kernel merely loads the component, but if it not a trusted component, the OS can choose not to load it (depending on Owner policy).
2. If the Owner permits this checking to be overridden, the OS must securely record a change in the level of trustworthiness of the OS. For example, the OS kernel would compute a digest of the untrusted component and of the Trusted Component Policy. It then concatenates the two digests with information that identifies the component and information that identifies the policy. This concatenated data is written as an event to the measurement log. The component signature and certificate (if they exist) would serve as identification of the component. Otherwise, the component identification information is whatever is available that might serve to identify the component. The name of the Trusted Component Policy might serve to identify the policy.
3. The OS kernel submits the event data to the TPM as the argument to an EXTEND operation on the OS Loader PCR.
4. The OS may choose to terminate or notify processes that the level of trust in the OS is about to change.
5. The OS kernel loads the new component.

Later, in replying to an integrity challenge, the TPM signs and returns the current OS Loader PCR. Any part of the Subsystem or platform returns the measurement log and information about the Trusted Component Policy. A Challenger may be satisfied to receive just a reference to a standard policy, but may require a verbatim copy of a non-standard policy.

The Challenger may use such information as exists in the measurement log in an attempt to determine the proper value of component digest (the information that should be in the component signature) and the *bona fides* of the component source (the information that should be in the component certificate). The verification process may include obtaining the proper value of component digest from the component source (or a proxy) and using a policy to verify that the component source is trusted by the Challenger. The Challenger can then use its own policies to decide whether, in the opinion of the Challenger, the component can be treated as a trusted component for the intended purpose.

Provided the Challenger has gathered sufficient information (from the measurement log and/or other sources), it can calculate a predicted value of the OS Loader PCR. If that predicted value matches the value reported by the TPM, then the Challenger may consider that the platform has sufficient trust for the intended purpose. Alternatively, a Challenger may insist that the platform reboot back to the original trusted state before the Challenger is willing to take part in interactions.

#### **2.11.6 Embedding a TPM in other security functions**

It may be advantageous to pass plaintext data from the TPM directly into non-TCPA security modules within the platform. This is desirable if the platform is partitioned into “red” areas (where data is in a vulnerable format) and “black” areas (where data is in an invulnerable format), for example.

TCPA therefore enables the passing of TPM plaintext output directly to other arbitrary functions by defining a TPM with multiple output ports, and permitting plaintext data protected (see section 2.8.2) by the TPM to be sent to any of those ports. Any plaintext output of commands using certain keys (tagged as “redirected” keys) is sent by the TPM directly to a chosen port, and is not interpreted by the TPM.

The use of multiple TPM output ports is optional. The use of data at such “redirected” ports is outside the scope of this specification.

## **2.12 Data Structures**

This section presents a summary of some data structures that enable the TPM to operate and be managed. A TPM is controlled using authorized commands. The method of generating authorization data is outside the scope of this specification.

### **2.12.1 Credential Descriptions**

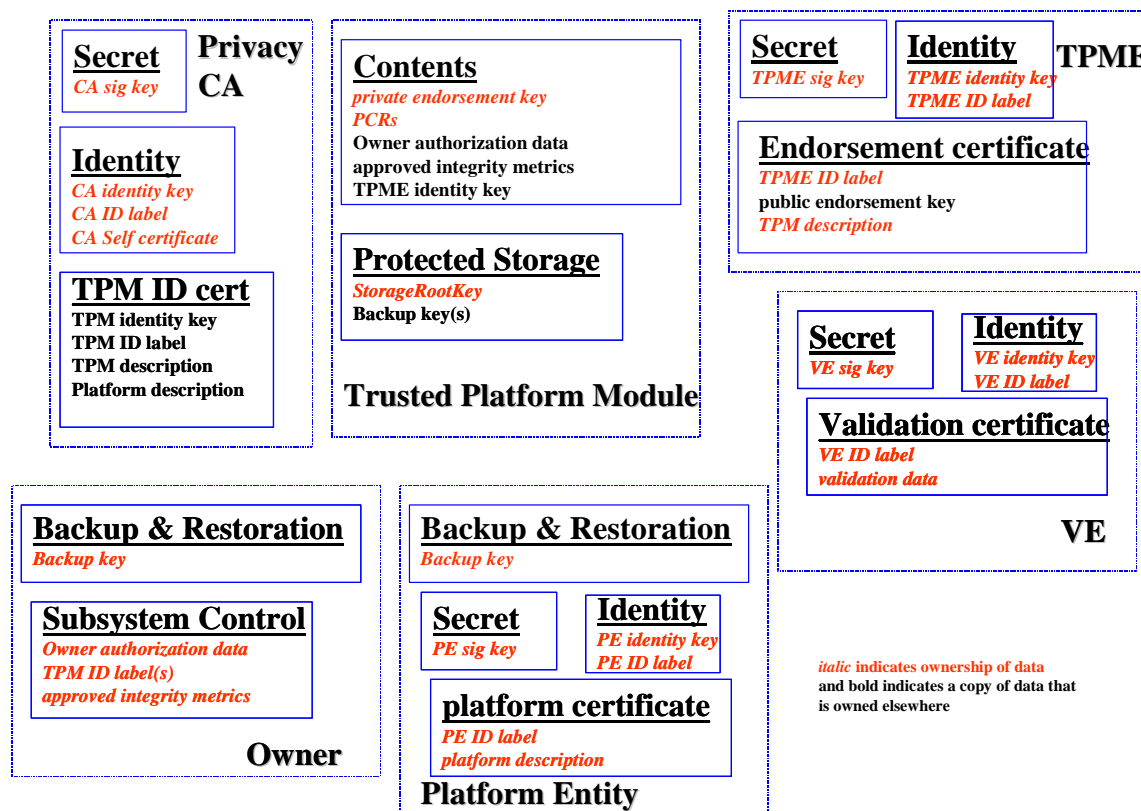
A Subsystem or its associated platform may store certificates. These are not essential for the Subsystem itself, but are useful because of their operational advantages when replying to integrity challenges. An Integrity Challenger requires the data in these certificates in order to judge the validity of integrity metrics measured in the platform. So, receiving the data from the Subsystem relieves the Challenger of the need to fetch it independently. In the future, the data may enable the Subsystem to generate an integrity summary of the measured metrics that matched the proper values (and of those that did not) for a Challenger with restricted computational power.

These certificates are as follows:

- The “validation certificate,” which may be stored anywhere on the platform. Each Validation Entity produces such a certificate, which vouches for part, parts or all of a platform and is normally installed during the same process that installs the part endorsed by the VE. Each platform of the same type may have the same validation certificates. Each validation certificate is signed by the VE and contains data that includes the “TCPA Validation Data” statement, the VE identity label and the proper (predicted) value of the integrity metric.
- The “endorsement certificate,” which may be stored anywhere on the platform. If the certificate is stored on the platform, it should be stored in a form that has restricted access so as to protect the privacy of the Owner of the TPM. The TPME produces this certificate, which vouches for the existence of a genuine TPM. This certificate is (statistically) unique to each TPM. Each endorsement certificate is signed by the TPME and contains data that includes the statement “TCPA Trusted Platform Module Endorsement,” the TPM’s public endorsement key, the TPME’s identity label and a description of the TPM.
- A “TPM-identity certificate,” which may be stored anywhere on the platform. If the certificate is stored on the platform, it should be stored in a form that has restricted access so as to protect the privacy of the Owner of the TPM. This is produced by a (Privacy) CA chosen by the Owner and vouches for the identity of a Subsystem. This certificate is (statistically) unique to each Subsystem. Each Subsystem-identity certificate is signed by the CA and contains data that includes the statement “TCPA Trusted Platform Module identity,” the TPM identity public key, the Subsystem identity label, a description of the TPM and the platform
- A “conformance certificate,” which is produced by a Conformance Entity chosen by the manufacturer of the platform to vouch that the platform design meets the requirements of this specification. Each conformance certificate is signed by the CE and contains identifiers that indicate the strength of protection that is afforded by the platform.
- A “platform certificate,” which may be stored anywhere on the platform. If the certificate is stored on the platform, it should be stored in a form that has restricted access so as to protect the privacy of the Owner of the TPM. This is produced by the manufacturer of the platform in order to attest that the platform has been built according to an accredited design. Each platform certificate is signed by a PE and contains data that includes the statement “TCPA Trusted Platform Endorsement,” a reference to

the specific TPM built into the platform, a reference to the RTM built into the platform, a reference to the appropriate conformance certificate, and a reference to the appropriate endorsement certificate. A number of ordinary certificates also may be required. These are conventional certificates attesting to the identity of entities such as the TPME and the Validation Entities.

## 2.12.2 Data Descriptions



Each TPM uses a private endorsement key. This key, in conjunction with the endorsement certificate, proves that the TPM is genuine.

The TPM must store the measured values of the platform's integrity metrics in the PCRs.

A TPM with just an identity cannot, however, be managed. This is because it cannot verify any commands from the organization or individual that owns it. Each TPM therefore requires "Owner-authorization data" that identifies the Owner. This data is known only to the TPM and the Owner.

Each TPM supports protected storage. This requires the StorageRootKey (SRK) and at least two backup keys. The TPM uses protected storage to hold TPM identity information and information submitted to the TPM. (Protected storage is not shown on the preceding diagram.)

A TPM identity has three parts:

1. The "TPM ID label," a number or textual label that acts as a conventional ID. The TPM ID label will ordinarily be used as an ID of the platform associated with the Subsystem.
2. The "TPM identity key," a cryptographic identity of the TPM. The TPM identify key will ordinarily be used as a cryptographic identity of the Platform.
3. The "TPM identity signature key," which proves a TPM's identity by signing data sent from the TPM.

### 2.12.3 Data Initialization and Maintenance

The following description is non-exhaustive. The precise manner of generation and storage depends on the exact situation, so consider the examples here as only a guide.

#### 2.12.3.1 Random Number Generator

A pseudo-random generator is initialized with a seed from a non-deterministic source. The entropy of the generator is condensed and whitened by passing  $n$  bits of the output of the generator through a cryptographic hash that produces  $h$  bits of output. If the non-deterministic source has entropy  $e$ , it is required that ( $n \geq h/e$ ). At any time, further non-deterministic data may be mixed into the pseudo-random generator.

#### 2.12.3.2 Owner

The Owner must authenticate his or her commands to a TPM. This is done using authorization data. (The method of generating that data is outside the scope of this specification.) The authorization data must be kept safe and secret so that it is known only to the Owner and to the TPM.

When a TPM is given an identity, the Owner loads the TPM with a number or other label that acts as a conventional ID of that TPM. The TPM ID label is loaded into the TPM by the Owner, using a command authorized by the Owner.

If the Owner is to perform TPM backup and recovery, a public key is required. The Owner can insert this into the TPM using a command authorized by the Owner.

#### 2.12.3.3 TPM

The TPM can generate asymmetric keys pairs so that the private key never need be exported outside the TPM. The private key must be kept safe and secret, while the public key need only be kept safe. The TPM contains a private endorsement key that is used to prove that the TPM is actually a TPM. The corresponding public endorsement key is exported from the TPM to the TPME, which uses it to produce an endorsement certificate.

To enable protected storage, the SRK must be created inside the TPM. The Owner-authorization data enables the TPM to be customized. It enables control of the TPM, and (in particular) the generation of TPM identities. Loading of the Owner-authorization data key must be done reliably and is preferably done as soon as possible so as to personalize the TPM. This “locks” the TPM and minimizes the opportunity for misuse of stolen or mislaid platforms. Security is enhanced if there is a different Owner-authorization data for each TPM.

A TPM's identity has a public asymmetric key. The corresponding private asymmetric key, a TPM signature key, enables the TPM to sign data and therefore to prove its identity. The TPM identity public key must be kept safe and protected, for privacy rather than security reasons. The TPM signature key is *never* revealed outside the TPM. There *must* be at least one identity/signature key pair for each Subsystem before it can take part in an integrity challenge/response. Measured integrity metrics are obtained by the Subsystem in the manner described in the specification.

#### 2.12.3.4 TPM Entity

The TPME vouches for the fact that a TPM is actually a TPM. The TPME causes a TPM to generate an endorsement key pair in a “clean” environment. It constructs and signs the endorsement certificate. The TPME must be the entity responsible for ascertaining that the TPM is genuine and that it has generated a particular endorsement key. The TPME might be the TPM manufacturer or the platform OEM, for example.

The TPME loads the TPM with the TPME's identity key. This enables upgrades of capabilities by the TPME.

### 3. Glossary

This section briefly describes the new terms that are introduced in this document.

**Authenticated Boot:** the normal boot process of a Trusted Platform; a boot process where measurements of the boot process are reliably measured and stored.

**Challenger:** a person or other entity that wishes to determine whether a target platform can be trusted for a particular intended purpose.

**Conformance Entity:** a person or other entity that vouches that the design of a platform satisfies the requirements of this specification.

**Conformance Certificate:** a certificate signed by the Conformance Entity that attests that the design of a platform satisfies the requirements of this specification.

**Endorsement Certificate:** a certificate signed by the Trusted Platform Module Entity that contains the public key of a genuine set of trusted capabilities (a Trusted Platform Module).

**Endorsement key:** a “public endorsement key” is the public key of an asymmetric pair that identifies a genuine set of trusted capabilities; a “private endorsement key” is the private key of an asymmetric pair (known to a genuine set of trusted capabilities, only).

**EXTEND:** a trusted capability that appends new integrity data to old integrity data, such that a summary of both data and the order of data are preserved: a concatenation of a new integrity metric and an existing Platform Configuration Register, followed by a hash, and the loading of the resultant digest in the PCR.

**Integrity metric:** data that is a condensed value of information that indicates the history of the platform, in so far as that history affects the trustworthiness of the platform.

**Owner:** the administrator of the trusted capabilities inside a Trusted Platform.

**Platform Entity:** a person or other entity that vouches that a platform was built to an accredited design that satisfies this specification.

**Platform Certificate:** a certificate signed by the Platform Entity, which attests that a platform containing a particular Trusted Platform Module is a genuine Trusted Platform.

**Platform Configuration Register:** a memory location that is shielded from unauthorized intrusion, and contains data that is a condensed value of information that indicates the history of the platform, in so far as that history affects the trustworthiness of the platform.

**Privacy-CA:** a Certification Authority selected by the Owner of a Trusted Platform to attest to an identity of a trusted platform while maintaining the privacy of the Owner.

**Protected Storage:** a set of trusted capabilities that provide a portal to stored data, such that data cannot be interpreted unless the correct portal and correct authorization data are used.

**QUOTE:** a trusted capability that returns a signed value of the data in a Platform Configuration Register; the means of obtaining a signed copy of condensed integrity measurements.

**Root-of-Trust-for-Measurement:** a collection of capabilities that must be trustworthy if measurements of the environment inside a platform are to be trusted.



**Root-of-Trust-for-Reporting:** a collection of capabilities that must be trustworthy if reports of measurements of the environment inside a platform are to be trusted.

**Root-of-Trust-for-Storage:** a collection of capabilities that must be trustworthy if (protected) storage of data inside a platform is to be trusted.

**SEAL:** a trusted capability that associates stored data with a stated value of Platform Configuration Register, such that the data cannot be retrieved unless the current value of PCR matches the stated value of PCR; a means of making a particular software environment part of the access controls for data.

**StorageRootKey:** a key at the top of a key hierarchy in (protected) storage of data.

**Secure Boot:** when a Trusted Platform checks the progress of its boot sequence against the expected boot sequence, and raises an exception if there is a difference.

**Subsystem:** the collection of capabilities inside a platform that enable the platform to be trusted.

**Support Services:** a set of capabilities inside a platform, which do not themselves need to be trustworthy, that are nevertheless required if the platform is to be trusted.

**Trust (definition):** an entity can be trusted if it always behaves in the expected manner for the intended purpose.

**Trusted Platform:** a platform that contains trusted capabilities and their support capabilities.

**Trusted Platform Module:** the set of trusted capabilities that are independent of the type of platform; all the trusted capabilities except the Root-of-Trust-for-Measurement; the specification does not prescribe any particular type of embodiment of a TPM.

**Trusted Platform Module Entity:** a person or other entity that vouches that a particular public (endorsement) key was generated by a set of genuine trusted capabilities (the Trusted Platform Module).

**TPM-identity Certificate:** a certificate signed by a (privacy) Certification Authority, which attests that a particular identity belongs to a genuine Trusted Platform.

**Trusted Services:** a set of capabilities inside a platform that must be trustworthy for the platform to be trustworthy.

**User:** a person or other entity that uses the services provided by a Trusted Platform.

**Validation Entity:** a person or other entity that vouches for part of a platform by attesting to the value that should be obtained if an integrity measurement is made on that part of the platform while it is working as intended.

**Validation Certificate:** a certificate signed by the VE that contains the value that should be obtained if an integrity measurement is made on a particular part of the platform while it is working as intended.