

PC Client Specific Virtualized Trusted Platform Specification

Specification Version .44

Revision 1

October 11, 2013

Draft

**Work In Progress**

This document is an intermediate draft for comment only and is subject to change without notice. Readers should not design products based on this document.

TCG CONFIDENTIAL

Copyright © TCG 2011

Copyright © 2011 Trusted Computing Group, Incorporated.

**Disclaimer**

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. Without limitation, TCG disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims 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.

No license, express or implied, by estoppel or otherwise, to any TCG or TCG member intellectual property rights is granted herein.

**Except that a license is hereby granted by TCG to copy and reproduce this specification for internal use only.**

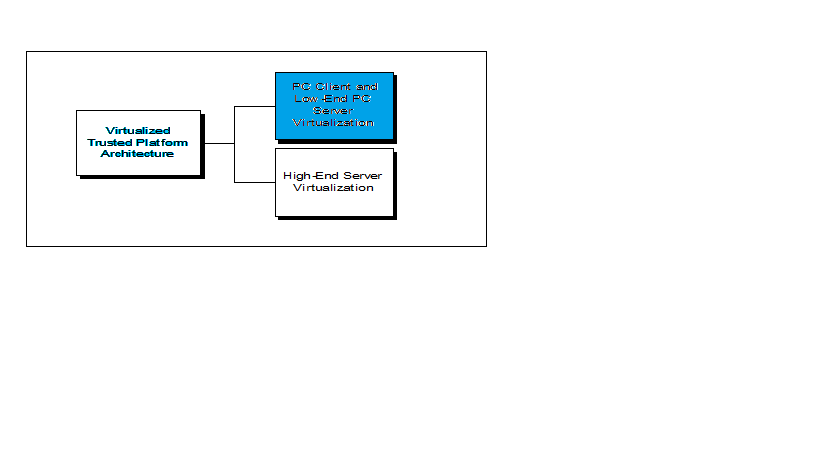
Contact the Trusted Computing Group at www.trustedcomputinggroup.org for information on specification licensing through membership agreements.

Any marks and brands contained herein are the property of their respective owners.

**Revision History**

|  |  |
| --- | --- |
| 01/03/11 | Reset of document, based on the final high level Virtualization Specification |
| 4/7/11 | Change spec to reflect answers to migration received at meeting |
| 4/12/11 | Added in section on vTPM ownerAuth |
| 8/17/11 | Reformatted to be closer to ISO/IEC standard |
| 11/9/11 | Incorporated comments from D. Challener |
| 10/12/12 | Spec reorganized |
| 11/12 | Normative requirements established, distinct from informative text |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**VPWG Document Roadmap**



**Acknowledgements**

The VPWG wishes to thank all those who contributed to this specification.

|  |  |
| --- | --- |
| Asher Altman | Intel |
| Stefan Berger | IBM |
| Rene Bourquin | General Dynamics |
| Benoit Poulot-Cazajous | Bertin Technologies |
| David Challener | Johns Hopkins University |
| Conan Dailey | General Dynamics |
| Laszlo Elteto | SafeNet |
| David Gilbert | IBM |
| Kenneth Goldman | IBM |
| Carey Huscroft | Hewlett-Packard |
| Chris James | Johns Hopkins University |
| Dean Liberty (Co-Chair) | AMD |
| Francois Lucas | Thales Communications |
| Bob Malek | Unisys |
| Eugene Myers | US Government |
| Frank Molsberry | Dell |
| Florian Samson | BSI |
| Jose Sancho-Dominguez | Hewlett-Packard |
| Paul Sangster | Symantec |
| Vinnie Scarlata | Intel |
| Mark Scott-Nash | Intel |
| Ariel Segall (Editor) | MITRE Corporation |
| Leslie Stolp | HP |
| Chris S. Anderson | US Government |
| Lee Terrell | IBM |
| Lee Wilson (Co-Chair) | IBM |
| Kenneth Goldman | IBM |
| Monty Wiseman | Intel |

Table of Contents

1 Introduction 11

2 Scope (informative) 12

2.1 Goals 13

2.2 Current Non-Goals 13

2.3 Intended Future Goals (Not Addressed in this Version) 14

2.4 Target Trusted Platform Functions 14

3 References (informative) 16

4 Terms and Definitions 18

5 Keywords and Conventions (normative) 21

6 Design Principles (informative) 22

6.1 The Need for Virtualized Trusted Platforms 22

6.2 Virtualization of a Trusted Platform 22

6.3 vTPM Design Goals 23

6.4 Assumptions 25

6.5 Use Cases (informative) 25

6.5.1 When Virtualized Trusted Platforms are Useful **Error! Bookmark not defined.**

6.5.2 System with Multiple VMs and Simple Virtual Platforms 26

6.5.3 System with Complex Virtual Platforms 27

6.5.4 System with Virtual Platforms and Shared Helper Domains 28

6.6 Security Considerations (informative) 29

6.6.1 Threat Model 29

6.6.2 Countermeasures 30

7 Hardware Platform Support 34

7.1 Initial pTPM provisioning goals (normative) 34

7.2 pTPM Configuration Requirements (normative) 34

7.2.1 pTPM state 34

7.2.2 pTPM keys and certificates 34

7.2.3 pTPM monotonic counters **Error! Bookmark not defined.**

7.2.4 pTPM Certification 36

*7.3* Non-pTPM Provisioning Requirements 36

7.3.1 Boot sequence 36

7.3.2 Security Launch Policies (normative) 36

7.3.3 IT Authority Keys (normative) 36

7.3.4 Protecting IT Authority Keys (informative) 37

7.3.5 Procedure for Initial pTPM Provisioning (informative) 37

7.4 FRU Replacement (Optional) **Error! Bookmark not defined.**

7.4.1.1 Prepare Field Replaceable Unit for Future Replacement **Error! Bookmark not defined.**

8 Software Architecture Components 39

8.1 Virtual Machine Manager (VMM) 39

8.1.1 VMM Requirements (normative) 39

8.1.2 VMM Description (informative) 40

8.2 vPlatform Manager 41

8.2.1 vPlatform Manager Requirements (normative) 41

8.2.2 vPlatform Manager Description (informative) 42

8.3 vTPM Factory 43

8.3.1 vTPM Factory Requirements (normative) 43

8.3.2 vTPM Factory Description (informative) 43

8.4 vTPM 44

8.4.1 vTPM Requirements (normative) 44

8.4.2 vTPM Role (informative) 44

8.5 Virtual Platform (VP) 45

8.5.1 Virtual Platform Requirements (normative) 45

8.5.2 Virtual Platform Role (informative) 46

8.6 Migration Authority 48

8.6.1 Migration Authority Requirements (normative) 48

8.6.2 Migration Authority Discussion (informative) 48

8.7 vRTM 49

9 Platform Operation 51

9.1 Booting and Measuring the System 51

9.1.1 System Boot Context (informative) 51

9.1.2 Requirements for System Boot (normative) 51

9.2 Creating a New Virtual Platform 51

9.2.1 Virtual Platform Creation Context (informative) 51

9.2.2 Requirements for Virtual Platforms (normative) 52

9.2.3 Requirements for Newly Initialized vTPMs (normative) 52

9.3 Protecting vTPM Data 52

9.3.1 vTPM Data Protection Context (informative) 52

9.3.2 vTPM Data Protection Requirements (normative) 54

9.3.3 Sealing vTPM secrets to PCRs (informative) 54

9.3.4 Binding secrets to a key held by a third party (informative) 55

9.3.5 Hybrid Approach (informative) 56

9.3.6 Binding secrets to an agent (informative) 56

9.4 Launching a Previously Executed vPlatform 57

9.4.1 vTPM Launch Requirements (normative) 57

9.4.2 Approaches to Launching a vTPM (informative) 57

9.4.3 Launching a vTPM using PCR-sealed data (informative) 58

9.4.4 Launching a vTPM using remotely protected data (informative) 59

9.4.5 vTPM PCR Contents on vPlatform Launch 59

9.5 Runtime Operation 60

9.5.1 Runtime Operation Considerations (informative) 60

9.5.2 Runtime Operation (normative) 61

9.6 Saving vPlatform State 61

9.6.1 State Saving Requirements (normative) 61

9.6.2 Saving State Examples (informative) 62

9.7 vPlatform Shutdown 64

9.7.1 Shutdown Requirements (normative) 64

9.7.2 Shutdown Discussion (informative) 64

9.8 Backing Up and Restoring vPlatforms 65

9.8.1 Backup and Restoration Requirements (normative) 65

9.8.2 Backup and Restoration Discussion (informative) 66

9.8.3 Backups using trusted server key (informative) 66

9.8.4 Rollback Prevention Discussion (informative) 66

9.9 Software Upgrade 67

9.9.1 Upgrade Requirements (normative) 67

9.9.2 Upgrade Considerations (informative) 67

9.10 pPlatform Change (Migration) 69

9.10.1 pPlatform Change Requirements (normative) 69

9.10.2 pPlatform Change Discussion (informative) 69

9.10.3 Migration and Backup Techniques (informative) 70

9.10.3.1 Local Owner Approval 70

9.10.3.2 Using a remote server to approve migrations 71

9.10.3.3 Using a local backup server 71

10 Attestation and Remote Verification 73

10.1 The Importance of Remote Verification (informative) 73

10.2 vTPM Certification Requirements (normative) 73

10.3 vTPM Certification Discussion (informative) 74

10.4 Virtual Platform Credentials 75

10.4.1 Virtual Platform Credential Requirements (normative) 75

10.4.2 Virtual Platform Credential Discussion (informative) 76

10.5 Attestation of Virtual Platform State 76

10.5.1 Virtual Platform Attestation Requirements (normative) 76

10.5.2 Virtual Platform Attestation Discussion (informative) 77

10.5.3 Virtual Platform RTMs (informative) 78

10.5.4 Simple Virtual Platform Software Attestation (informative) 78

10.5.5 Complex Virtual Platform Software Attestation (informative) 78

10.6 Deep Attestation 79

10.6.1 Deep Attestation Requirements (normative) 79

10.6.2 Deep Attestation Example Implementations 80

10.6.2.1 Network-Visible Attestation Manager 80

10.6.2.2 Deep Quote Command Description (informative) 81

10.6.2.3 Deep Quote Command Requirements 81

10.6.2.4 Ephemeral AIK Description (informative) 82

10.6.2.5 Ephemeral AIK Requirements (normative) 83

10.6.3 Privacy and Deep Attestation (informative) 85

# Introduction (informative)

The TCG VPWG (Virtualized Platform Work Group) provides a high level architecture [10] which describes various methods for managing the security of virtualized platforms. This high level architecture defines terms, discusses general problems encountered in virtualized platforms, and discusses general techniques for addressing those problems. The high level architecture document is not intended to be an implementation guide; to assist manufacturers with specific implementations considered to be of primary interest to the industry, the VPWG provides deployment scenario auxiliary specifications. This document describes one of those deployment scenarios: x86 Client and Stand Alone Server for TPM 1.2. We assume that both the hardware TPM and virtual TPMs are using the 1.2 specification although accommodations for TPM 2.0 have been made; future versions of this specification may use TPM 2.0.

This PC Client specific version of the Virtualized Trusted Platform specification addresses clients and low-end servers such as SOHO (Small Office, Home Office) servers. Future versions of this specification or related specifications may expand into full data center virtualized platforms and alternative platforms such as mobile or embedded virtualized platforms. The platforms most likely to use the PC Client version of the specification are the same ones that are likely to use the TCG PC Client Specific Implementation Specifications for building hardware, although future versions may support broader architectural choices.

# Scope (informative)

This specification is intended for developers of systems providing trusted virtual environments based on client or low-end servers. The virtual environments are not expected to migrate, between different physical platforms during normal use. This specification does provide for replacement of physical components, including the physical platform TPM, for repair or upgrade. It also provides for offline migration of the all the virtual environments *en masse* to a different physical platform, as this is equivalent to replacement of the underlying physical platform parts.

This specification does not describe in detail the launch and measurement of a trusted VMM . Instead, it assumes that such an implementation is based on the TCG PC Client specifications, including possible use of the DRTM specification.

This specification is, to the best of the writers’ ability, implementation-independent; it provides a set of requirements and recommendations which, if met, should allow a system to support trusted virtualization use cases.

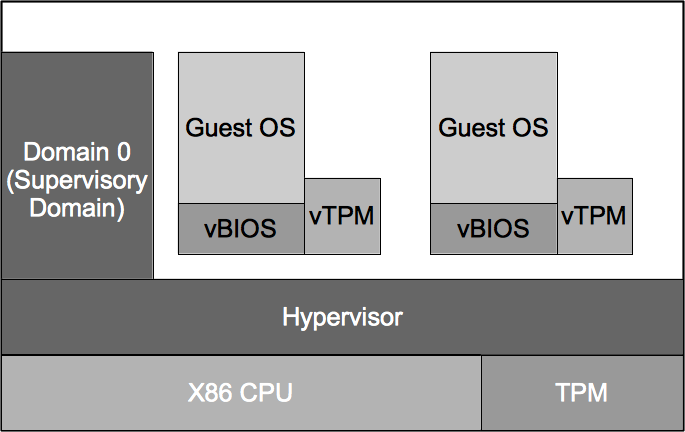


Figure – An Example x86 Client / Low End, Stand Alone Server Deployment Scenario

## Goals (informative)

The goals of this specification are to:

* Document the design considerations of a virtualized trusted platform built on the following TCG platforms:
* PC Client Platform
* Document some deployment considerations of a virtualized trusted platform on a PC Client Platform that affect the design of a trustworthy system. These considerations will be largely applicable to other platform architectures, but the focus of this specification will be on the PC Client.
* Identify and describe the primary components in the system,and how they are used in common client and low-end server operations.
* Discuss responsibilities and security considerations for each of the identified components.
* Identify detailed requirements and recommended approaches for each component when appropriate. This specification follows the guidelines identified in the Virtualized Trusted Platform Architecture Specification [10].
* Minimize the number of changes to existing TCG technologies required to support virtualized trusted platforms. In particular, we should provide the necessary components to support TNC at the level of the virtual platform, and should support vTPMs consistent with the TPM specification.
* For simplicity, this specification assumes a single VMM is running on the platform (not nested VMMs). The concepts provided in this specification can be extended to nested VMMs and this use is not precluded.

## Current Non-Goals

The following lists several of the key non-goals or out of scope items for this version of this specification. Such things may be provided by an implementation that meets this specification, but this specification will not try to provide a means of solving these problems.

* Identifying specific data necessary to describe the security state of the VMM or VMs.
* This data is both implementation and appraiser specific, and too detailed for a general-purpose specification. It is likely that TNC, PTS, SCAP, etc. specifications will be relevant to implementations.
* This specification does provide requirements for key system information that the chosen data must cover.
* Para-virtualization (i.e. use of physical TPM to provide functionality of multiple “virtual” TPMs.)
* Non-trusted Virtual Platforms (Virtual Platforms without vTPMs)

## Intended Future Goals (Not Addressed in this Version)

The following lists several future goals of the VPWG which are not addressed in this version of the specification. The plan is to address these in future versions of this specification or other VPWG specifications.

* Live Migration of a Virtual Platform
* Migration Authority details
* Canonical description of TPM 1.2 and 2.0 TPM contents, for purposes of migratable and/or interoperable vTPMs.
* Hot standby, in which an identical backup VM (including vTPM) is running in parallel with the primary copy so as to allow rapid switching in the event of a failure
* Hibernation and suspension of the physical platform that is supporting virtual platforms.

## Target Trusted Platform Functions

The goal of this specification is to provide descriptions of and support for the following functions:

* Initial physical TPM (pTPM) provisioning Platform Startup (including pTPM)
* Prepare field replaceable unit (FRU Stock)
* Virtual platform preparation and usage
* Initial provisioning of vTPM
* Binding a virtual TPM (vTPM) to a VM
* Establishing initial measurements in vTPM PCRs during VM startup.
* Secure linkage between a VM and its underlying VMM via the vTPM.
* Secure storage of vTPM secrets using pTPM
* Anti-rollback of vTPM data
* Unique execution of each vTPM (i.e. only one active copy at a time)
* Use of a vTPM to identify a Virtual Platform
* Trusted attestation for VMs (e.g., extending certificate chain)
* Re-provisioning (VMM upgrade)
* vTPM backup
* vTPM move (e.g., maintenance and upgrade of system hardware)
* vTPM upgrade (e.g., new hashing algorithm)
* vTPM startup
* vTPM shutdown, orderly and abrupt
* vTPM suspend
* vTPM destruction
* vTPM reset

# References (informative)

For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

1. Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”, Internet Engineering Task Force RFC 2119, March 1997. <http://www.ietf.org/rfc/rfc2119.txt>
2. Trusted Computing Group, TCG Credentials Profiles Specification Version 1.1, <http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_tcg_credential_profiles_specification>, May 2007.
3. Trusted Computing Group, *TNC Architecture for Interoperability*, Specification Version 1.0, <http://www.trustedcomputinggroup.org/resources/tnc_architecture_for_interoperability_specification>, April 2005.
4. Trusted Computing Group, TPM Main Specifications, Version 1.2 <http://www.trustedcomputinggroup.org/resources/tpm_main_specification>.
5. Trusted Computing Group, Platform Trust Services Interface Specification (IF-PTS), <http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_platform_trust_services_interface_specification_ifpts_version_10>, November 2006.
6. Trusted Computing Group, TNC Architecture for Interoperability, <http://www.trustedcomputinggroup.org/resources/tnc_architecture_for_interoperability_specification>, May 2009.
7. Trusted Computing Group, TCG Architecture Overview, <http://www.trustedcomputinggroup.org/files/resource_files/AC652DE1-1D09-3519-ADA026A0C05CFAC2/TCG_1_4_Architecture_Overview.pdf>, August 2007.
8. Trusted Computing Group, PC Specific Implementation Specification, <http://www.trustedcomputinggroup.org/resources/pc_client_work_group_pc_specific_implementation_specification_version_11>, August 2003.
9. Trusted Computing Group, TNC IF-MAP Binding for SOAP Specification, <http://www.trustedcomputinggroup.org/resources/tnc_ifmap_binding_for_soap_specification>, July 2010.
10. Trusted Computing Group, Virtualized Trusted Platform Architecture Specification, <http://www.trustedcomputinggroup.org/resources/virtualized_trusted_platform_architecture_specification>, September 2011.
11. Trusted Computing Group, TCG EFI Platform Specification Version 1.20, <http://www.trustedcomputinggroup.org/resources/tcg_efi_platform_specification_version_120_revision_10>
12. Trusted Computing Group, TCG Design, Implementation, and Usage Principles, <http://www.trustedcomputinggroup.org/resources/tcg_design_implementation_and_usage_principles_best_practices>
13. Trusted Computing Group, PC Client Work Group Specific Implementation Specification for Conventional BIOS Specifiation, Version 1.2, <http://www.trustedcomputinggroup.org/resources/pc_client_work_group_specific_implementation_specification_for_conventional_bios_specification_version_12>
14. Trusted Computing Group, PC Client Specific TPM Interface Specification (TIS), <http://www.trustedcomputinggroup.org/resources/pc_client_work_group_pc_client_specific_tpm_interface_specification_tis>
15. Trusted Computing Group, Registry of Reserved TPM 2.0 Handles and Localities, <http://www.trustedcomputinggroup.org/files/static_page_files/10437541-1A4B-B294-D0CBCDA0D8979C91/Registry of Reserved TPM 2.0 Handles and Localities_v1.0r0.pdf>
16. Trusted Computing Group, D-RTM Architecture Specification, <https://www.trustedcomputinggroup.org/resources/drtm_architecture_specification>
17. Trusted Computing Group, Trusted Platform Module Library Specification, Family “2.0”, <http://www.trustedcomputinggroup.org/resources/tpm_library_specification>

# Terms and Definitions

The following table lists the terminology used throughout this document. These terms are in addition to those terms defined in the Virtualized Trusted Platform Architecture Specification.

|  |  |
| --- | --- |
| **Name** | **Description** |
|  |  |
| Helper VM | This is a virtual machine which is developed to isolate functionality sometimes done in the primary OS or in the VMM, so as to provide smaller subsystems for security analysis; or which provides isolated functionality which provides a service to the primary VM in its vPlatform.. A helper VM might run a vPlatform Manager, separating its functionality from the VMM or supervisory domain. A different kind of helper VM might use VM introspection to take runtime measurements of the primary OS kernel, or provide a virtual CD drive. A helper VM or “service virtual machine” is analogous to a Unix daemon. While vTPMs could be described as helper VMs, vTPMs are sufficiently specialized that we do not generally use the term to refer to them. |
| Helper vPlatform | A vPlatform which provides services to a number of different vPlatforms on the same system. For example a virtual networking switch might be implemented as a helper vPlatform. Some systems may provide shared services as VMM applications instead; however, helper vPlatforms can provide additional isolation. |
| Measurement | A value which provides information about the identity and/or state of a particular piece of software. Although the most common measurement is a hash of a file, executable, or region of memory, other forms of measurement are possible. For example, software version information is an easily forged but less fragile measurement, while a kernel structure analysis may produce more meaningful information about OS state than a simple hash. Measurements are normally stored in TPM PCRs for future use in attestations. |
| Backup Server | This server is used to hold information useful for recovery and/or backup of a system. |
| Migration Authority | A party—normally a trusted third party—which is responsible for approving the migration, backup, and restoration of vTPMs. (All three functions are included because they are effectively the same operation with different policies.) |
| Migration Engine | A component of a trusted platform which is responsible for the local operations supporting migration, backup, and restoration, such as serialization and encryption of data. |
| Ephemeral AIK (eAIK) | An owner evict Attestation Identity Key which exists only while the associated vTPM and pTPM are bound together, which may be used for deep attestation. |
| BIOS | The term “BIOS” is used as a shorthand reference to refer to both legacy BIOS and UEFI firmware. It is also used to describe “boot firmware” in platforms that do not utilize legacy BIOS or UEFI firmware. (Note: BIOS is a term used extensively in the TCG PC Client Specification.) |
| Virtual Platform (vPlatform) | A group of one or more Virtual Machines working together to provide the services that might normally be provided by a hardware platform. The smallest trusted Virtual Platform consists of a primary (usually user OS) VM and a vTPM. |
| pTPM | Physical TPM |
| vPlatform Manager | A vPlatform Manager encompasses the functions of managing the vTPM, the subset of which is sometimes informally referred to as “vTPM manager”. Implementations may include a separate vTPM manager, but these remain under the vPlatform Manager architectural component definition.) |
| Trusted Computing Base  (TCB) | The Trusted Computing Base is the set of software which is both measured into the pTPM and relied upon to protect the secrets and integrity of other system components.  The VMM is the most critical component of the TCB, but other components such as the vPlatform Manager, Migration Engine, or shared cryptographic libraries whose compromise could result in a loss of secrecy for other well-behaved components are also included. Some systems may contain TCB components not listed in this specification. |
| VFK | vTPM Factory Key. A non-migratable pTPM signing key used to sign vECs. The equivalent of a TPM manufacturer’s key used to sign physical TPM Endorsement Credentials. |
| vEC | vTPM Endorsement Credential, certifying that a vTPM Endorsement Key belongs to a legitimate vTPM |
| VPCK | Virtual Platform Certification Key. A non-migratable pTPM signing key used to issue vPCs. The equivalent of an OEM’s key used to sign physical platform Platform Credentials. |
| vPC | Virtual Platform Credential, certifying that a vPlatform meets a particular Virtual Platform Specification |
| Virtual Platform Specification | The equivalent of a standard Platform Specification for a vPlatform, describing the vTPM feature requirements, expected vPCR usage, and other vPlatform properties. |
| VPMDK | vPlatform Manager Data Key. A non-migratable pTPM storage key used to encrypt the vPlatform Manager data. |
| VDK | vTPM Data Key. A symmetric key used to encrypt vTPM data when the vTPM is not running. |
|  |  |
| pPCRs | Platform Configuration Registers in the physical TPM |
| vPCRs | Platform Configuration Registers in a virtual TPM | |
| Appraiser | A remote entity which wishes to establish trust in the Trusted Virtualized Platform or a component Virtual Platform, using remote attestation. The appraiser decides whether it can trust the platform and for what uses based on TPM and vTPM evidence, along with any other evidence, such as TNC resports, which it may collect. | |

# Keywords and Conventions (normative)

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 (Refer to <http://www.ietf.org/rfc/rfc2119.txt>). This specification does not distinguish blocks of informative comments and normative requirements. Therefore, for the sake of clarity, note that lower case instances of must, should, etc. do not indicate normative requirements.

In order to improve the requirements visibility, each normative requirement is indented and placed in its own paragraph. Normative text has a white background; informative comments have a grey background.

# Design Principles (informative)

## The Need for Virtualized Trusted Platforms

In many use cases today, ranging from multi-level security workstations to cloud providers to mobile phones and even cars, trusted computing offers needed solutions to important security problems.. Reliable machine identity, protection of cryptographic material, and reporting of machine state are all valuable principles well beyond the original PC client scenarios. However, in many of these scenarios, merely proving the identity or trustworthiness of a VMM is insufficient. We wish to know whether the virtual machine we seek to use is reliable, whether the application wishing to download data is legitimate, or if critical software in the VM has been compromised. In these scenarios, bringing trusted computing principles and techniques into virtualized architecture is highly advantageous. By creating Virtual Platforms as an abstraction of hardware platforms in virtualized systems, providing them with virtual TPMs to support them, and tying our trust in those virtual TPMs down to the underlying hardware TPM, we can create multiple layers of trusted computing support that meet the needs of users and applications in a wide variety of use cases. Virtual TPMs (vTPMs) can attest to the state of a Virtual Platform’s component virtual machines, provide long-term identities that remote parties can use to identify the Virtual Platform, and protect the user’s secrets when the Virtual Platform is not in use. With vTPM support, we can provide VMs with the same security features that hardware TPMs provide in a non-virtualized platform.

## Virtualization of a Trusted Platform

A Trusted Virtual Platform will always have a vTPM. An Untrusted Virtual Platform would not have a vTPM (Note: these are not addressed herein). There may be a mix of these two running on a VMM.

This vTPM will appear to the guest operating system to be a TPM, in that software developed for a pTPM should function without change on the vTPM. The vTPM will comprise a virtual Root of Trust for Reporting (vRTR) and a virtual Root of Trust for Storage (vRTS) for the VM.

There must exist a virtual Root of Trust for Measurement (vRTM) responsible for populating PCRs in the vTPM reliably (in a trustworthy manner) in accordance with the PC Client specification when a VM is launched. Unlike the hardware RTM, the vRTM is not immutable. There are three known approaches to creating a vRTM (there may be others):

1. Run a virtual boot chain of trust using the same approach as a standard platform’s boot chain of trust to comply with PC Client specification
2. Have the (measured and trusted) VMM (or one of its components) take the initial vPlatform measurements and place them in appropriate vTPM PCRs
3. Have a (measured and trusted) helper VM create the initial vPlatform measurements and place them in appropriate vTPM PCRs. In this case, the helper VM must be measured into the pTPM PCRs and verified at vTPM launch.

All three virtual roots of trust (vRTR, vRTS, vRTM) must themselves be rooted in the corresponding hardware root of trust. The vRTR, responsible for providing a trustworthy mechanism for reporting on system state and certifying vTPM keys, is itself certified via a chain of cryptographic operations that originates in the pRTR. The vRTS, responsible for protecting the integrity and secrecy of data, is itself encrypted via a chain of keys originating in the pRTS. The vRTM, responsible for measuring the state of the Virtual Platform, is itself measured via a chain of measurement mechanisms originating in the pRTM. Each of them are part of a chain of trust that is externally verifiable by the mechanisms described in Attestation and Remote Verification (Section 10).

## vTPM Design Goals

In order to ensure that we have a virtual TPM that is trustworthy and can be used in the same manner as a physical TPM, rather than a mere TPM emulator which provides the same API, there are certain design goals and challenges that must be met. Many of these properties are provided for free by physical TPMs due to the simple fact of their physicality. (Physical objects cannot, for example, be freely copied.) Others have to do with the use cases for virtual TPMs. Note that the following are not requirements in a formal sense, but are rather motivating reasons for the requirements elsewhere in the specification.

1. **vTPM secrets should be protected from disclosure and tampering both when the vTPM is off, and when it is running.**
   * vTPM keys should only be accessible to the vTPM itself, or (when necessary) to measured components in the TCB. All vTPM data should be encrypted when the vTPM is off, and the vTPM must have some mechanism (usually enforced by the VMM) preventing untrusted components from accessing its secrets during operation.
2. **vTPMs should be unique.**
   * pTPMs are unique by their very nature, being physical hardware with internally generated random keys. vTPMs, being fundamentally software with data, are easy to copy. Mechanisms must be in place to prevent any vTPM from having two distinct versions, or from running in two places at the same time.
3. **vTPMs should contain verifiable information about the state of associated VMs**
   * Just as pTPMs store measurements of software running on the platform, so should vTPMs contain measurements of software running in the virtual platform. vTPMs are used by VMs in a very similar (almost identical) manner to the way it is done for pTPMs.
4. **vTPMs should be remotely verifiable**
   * Remote verifiers should have a way to determine how much trust they are willing to place in a vTPM, based on evidence.
5. **vTPMs should always be associated with vPlatforms meeting the same Platform Specification**
   * In particular, the PCRs of a given vTPM should have the same interpretation over the lifetime of the vTPM, as should the mapping between localities and particular other VMs in the virtual platform. Different platform specifications (eg. PC Client, Embedded, Mobile, etc.) populate PCRs in different ways; if a particular vTPM could be associated with different kind of platforms over its lifetime, there are security risks inherent in the interpretation of vTPM PCR contents or locality permissions. It is expected that virtual platforms will vary at least as much if not more than physical platforms because it is easier to create new configurations in a virtualized environment. Existing vTPMs cannot be associated with a virtual platform that does not meet the Platform Specification described in the vTPM’s credentials. A new vTPM must be manufactured if the Platform Specification changes.
   * While this requirement may often be implemented by maintaining a constant vPlatform to vTPM association, for simplicity and ease of use, that is not strictly required. In comparison, in the physical world, we would consider a pTPM to still be legitimate even if the pTPM’s motherboard were placed into a new machine with different hard drives and peripherals; we would not, however, consider it safe to put a pTPM meeting the Mobile specification into a machine expecting a PC Client TPM.
6. **vTPM trust should be rooted in a hardware TPM or equivalent secure hardware**
   * A “vTPM” that has no hardware root of trust is a TPM emulator and should not be considered trustworthy.
7. **vTPMs should be persistent through VM and platform reboots**
   * vTPMs accumulate data and state over the lifecycle of a vPlatform. This information must be kept for the lifetime of the virtual platform.
8. **vTPMs should never be rolled back to an earlier state without approval from an appropriate authority.**
   * There are a variety of attacks possible if the assumptions the TPM specification makes about time going forward do not hold. (For example, rolling back to a prior set of authorization values, or an earlier value of the monotonic counters.) As a general rule, rollbacks should not be done. There are special cases (eg. Failure cases, etc.) where it is required, but it should only be done when authorized by an approved authority.
9. **vTPMs should behave as pTPMs for software running in the VM**
   * Legacy software should be able to treat a vTPM exactly as if it were a pTPM without change, even if it does not take full advantage of vTPM trust architectures or features.
10. **vTPMs should have not have their performance limited by multiplexing a pTPM**
    * One of the major benefits of vTPMs is that software offers the possibility of a significant speedup over pTPMs. As a general rule, vTPMs should not rely on pTPMs for runtime operations as this will have severe performance implications.
11. **vTPMs should be isolated from untrusted components** 
    * Software has flaws; a component with as many trust requirements as a vTPM should be kept as isolated from attack vectors as possible.
12. **vTPMs should be able to have their software updated**
    * No software is perfect; we expect vTPMs and their trusted computing base (e.g. VMMs) to be updated. In most use cases, we expect the vTPM and its data to persist and still be usable after the update.
13. **All components of a Virtual Platform should migrate together**
    * A trusted Virtual Platform, consisting of a primary VM, a vTPM, and optional other components (usually helper VMs), is the virtual equivalent of a physical machine. It has a Platform Credential describing the components of the platform and their relationships, and the vTPM supports localities which are associated with specific components. Just as the vPlatform must always meet the Platform Specification which is described in the PC over many reboots, it must also continue to meet that specification when the platform migrates. Components described in the credential should migrate with the vTPM or have local equivalents on the destination machine, and the relationships between those components must match the PC.
14. **vTPMs should be capable of being backed up**
    * + - * In enterprise contexts, availability of data is critical; recovery from machine theft, hardware failure, or other loss may be required even if it requires compromising on security.

It is worth observing that several of these goals can require tradeoffs between them. For example, maximizing ties to hardware would make migration and high performance difficult, while backup capability is in tension with rollback prevention and uniqueness. Different use cases will emphasize different goals, and necessitate different choices in implementation.

## Assumptions (normative)

This specification makes the following assumptions about the architecture on which the virtualized platform is built:

The following virtualization technologies are used:

Hardware based processor virtualization

I/O remapping, particularly including DMA

The following platform components are used:

Physical TPM meeting the PC Client platform specifications. [14] or TPM 2.0 equivalent

System/BIOS meeting the PC Client platform specifications. [8] or TPM 2.0 equivalent

At least one RTM component is used, such as:

* + S-RTM [13]
  + D-RTM [1]

## Use Cases (informative)

This section discusses the use cases that provide a rationale for the features and requirements described in the remainder of the specification. Some of the described use cases are lower level than customer deployment level usages in order to provide the reader with an easier understanding of how the usage impacts the architecture and requirements defined within this specification.

Virtualized Trusted Platforms are most valuable when our use cases call for the advantages of virtualization technology, but the security features provided by trusted computing are also required. Virtualization’s advantages range from efficient use of hardware to ease of use for users to improved software monitoring. Trusted virtualized platforms can provide a cryptographically secure long-term identity for a virtual machine; protect secrets stored in our VM from unauthorized access; and determine the trustworthiness of a virtual machine before providing it with access to sensitive resources such as networks or files.

Trusted Virtualized Platforms should be used when:

* A remote party, ranging from a server of sensitive data to the VM’s owner, wants to confirm that they are communicating with the correct VM.
* A remote party wishes to have assurance that the VM they are communicating with is in a trustworthy state, or running approved software.
* A VM owner wishes to protect data stored in the VM from unauthorized access when the VM is not in use.
* A system owner wishes to ensure that sensitive data cannot be accessed unless the machine is in an approved configuration.

### System with Multiple VMs and Simple Virtual Platforms

In this scenario, a computer is used for business, banking, home, and travel. Different VMs are provided for each use case, with potentially distinct software installed in each one. VMs which will be used for secure transactions or to handle sensitive data will have an associated vTPM, forming simple Virtual Platforms; others may have no vTPM.

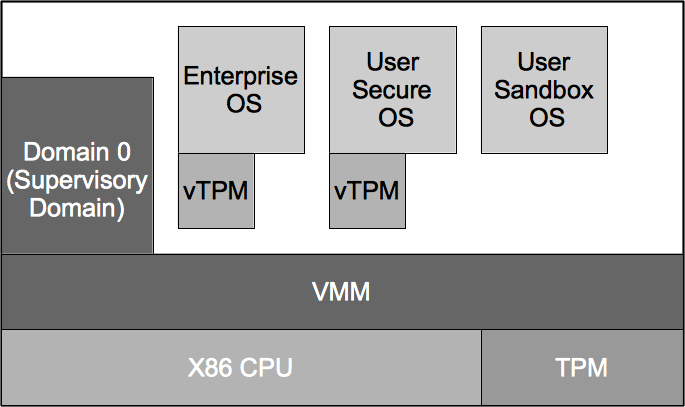


Figure : A system with several simple Virtual Platforms, some with vTPMs

For business use,, VMs are delivered by the enterprise; a vTPM is also provided, as part of the enterprise-approved virtual platform. The enterprise VM uses TNC to set up a VPN connection between the business and the VM. This uses attestation both from the pTPM and the vTPM to set up the connection, thus allowing the enterprise to determine that the system is in a secure state before allowing access to the intranet. Business secrets stored on the computer are secured by sealing encryption keys to the vTPM.

For personal use, or to access untrusted sites needed for work, a different VM is used., This VM can be used for games, email, web browsing, and other potentially unsecure activities, but will not be able to successfully attest itself to the enterprise to access business data.

For financial transactions or banking, there is a special, locked down VM, possibly provided by the financial institution, with its own vTPM. This VM is used to view or change financial information, perform e-banking, transfer funds, etc. The associated vTPM provides attestation both to the bank and to the user that the VM is in use and approved.

An alternate scenario using the same architecture is a shared computer used by a number of parties, such as in a hotel, college campus, or airport kiosk. A disposable VM uses attestation from its associated vTPM to provide the user’s smartphone with proof that the system is in a safe state to use. For ease of verification, these VMs are restored from a fresh image on each use, providing a standard set of user-verifiable measurements in the VM’s vTPM.

### System with Complex Virtual Platforms

In this scenario, which is most likely to occur in high-security situations such as multi-level workstations or Bring Your Own Device scenarios where entirely different environments controlled by different entities are desired, a system contains not only multiple user (also known as guest) VMs as described above, but utility VMs which provide specialized functionality to those guests. For example, a utility VM might provide a virtual device, or act as a specialized monitoring agent which assesses the health of the guest VM using VM introspection. These utility VMs should be grouped with their associated guest VM and isolated from each other, for maximal security and separation.

This is an example of a complex Virtual Platform; its closest equivalent is a hardware platform which contains both high-level functionality such as an OS and low-level support infrastructure such as the TPM and other devices. A Virtual Platform may consist of a single VM, or many. Virtual Platforms may also include components which are not VMs, such as processes running in shared service VMs.

In this scenario, a vTPM is one component within a Virtual Platform. Usually there will only be one vTPM per Virtual Platform, but we allow for the possibility of multiple vTPMs. The vTPM may be accessible by multiple VMs within the Virtual Platform; for example, a specialized measurement VM might place its measurement results into a vTPM PCR which the guest VM can then quote in response to external attestation requests.

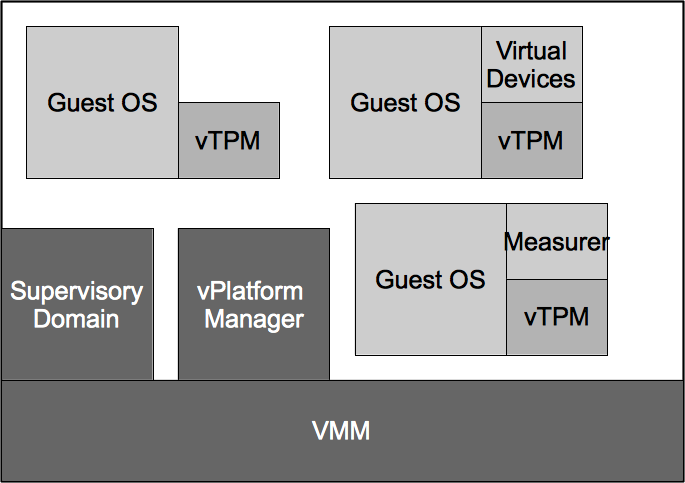


Figure : A system with several complex Virtual Platforms, consisting of multiple VMs associated with a single vTPM. Some VMs are special purpose, such as virtual devices or measurement domains, while others are general-purpose operating systems.

The above diagram shows an example Virtual Platform architecture. Each cluster of VMs separated by whitespace is a virtual platform; VMs within a virtual platform will usually have privileged access rights compared to VMs in other virtual platforms. For example, a guest OS will be able to talk to its own vTPM, but not to the vTPM of another virtual platform. Components in dark grey are the Trusted Computing Base; these components must be measured and verified in order to trust any platform component. Medium-grey VMs must be measured and verified in order to trust the individual vTPM, on which our trust in the virtual platform measurements will be built. Finally, the light grey domains are those which must be measured and verified in order to trust that particular VM.

Additional tiers of trust are possible—for example, a virtual platform may be set up so that trust in the guest OS relies on detailed measurements produced by a measurement domain, rather than via a hash of the guest OS image—but the basic idea remains the same.

### System with Virtual Platforms and Shared Helper Domains

This use case is very similar to the previous one; however, here, we have some specialized Virtual Platforms which are not guest operating systems run for a user. Instead, these Virtual Platforms are “helper” vPlatforms; they provide shared services for other VMs on the platform. In some architectures, these helper vPlatforms may provide single processes which are considered part of the guest vPlatforms; these processes might even include the vTPMs for other vPlatforms. In such cases, software running on the helper vPlatform would normally be included in the measurements of the guest vPlatform.

In other architectures, such as the one depicted below, the helper VPlatforms operate as independent Virtual Platforms with their own vTPMs. In these cases, external parties wishing to verify the trustworthiness of the system would have the option of including the helper vPlatform in their attestation, in order to confirm its correct operation.

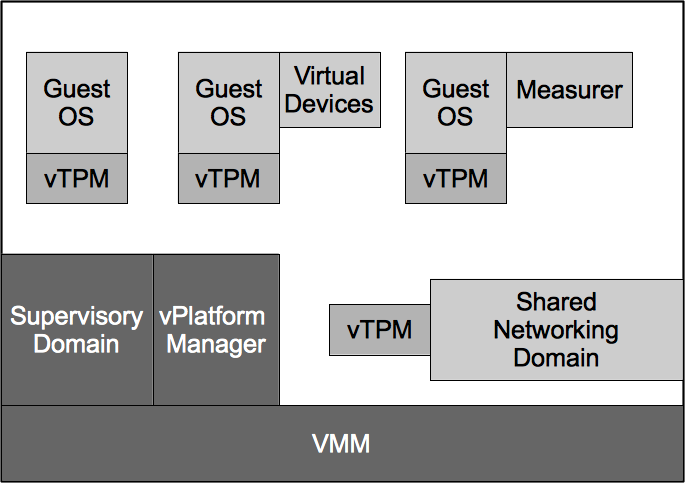


Figure : An example Trusted Virtualized Platform containing several guest Virtual Platforms and a helper Virtual Platform, providing shared access to the systems' single network card.

## Security Considerations (informative)

### Threats

#### Compromise of RTM firmware or settings/policy

* + Compromise of option ROM or settings

#### Compromise of vRTM firmware or settings/policy

* + Compromise of virtual option ROM or settings

#### Compromise of Boot Sequence

PXE, standard boot sequence stuff in clients etc.

#### Compromise of SMM

#### Compromise of VMM kernel

#### Compromise of vTPM Manager

Multiple copies of vTPMs could be running, vTPMs booting in a bad machine state, etc.

#### Compromise of the backup mechanism for vTPMs

#### Rollback of vTPM state (data, etc.)

#### Rollback of vTPM software

#### Loss of vTPM user privacy

Certs for vEK, Platform, vAIK, etc. have a great of identifying data.

#### Integrity (Change) of vTPM secrets and vPCRs

#### Confidentiality of vTPM secrets

#### Attack by a hostile VM running under the same VMM

#### DMA Attacks against VMM and VMs

#### Attacks on associations between VMs that share a vPlatform

Mis-association of VMs to vplatforms by the VMM, permissions are modified for VMs associated with a vplatform, each one may have a different locality which could be mis-assigned.

#### Attacks leading to multiple instantiations of a vTPM

#### Attacks leading to poor or no entropy seeding the generation of keys, etc.

#### Attacks on backed up, migrated or quiesced VM and VMMs

#### Attacks against integrity of attestation reports

#### Attacks against certificates used by the VMM, vPlatform, vPlatform Manufacturing Authority, etc.

#### Migration to an Attacker

#### Assigning Unwarranted Trust when Migrating from a Questionable Platform

### Countermeasures

The following countermeasures address the threats:

* Compromise of RTM firmware or settings
  + This is a threat against the physical platform. Our goal in this specification is to give vTPMs and vPlatforms the same security profile as pTPMs in physical platforms. PC Client Specification and the NIST SP 800-147 special publication.
* Compromise of vRTM firmware or settings
  + A compromise of a vRTM firmware or setting is detected by measurements made into the pTPM PCRs, which represent the configuration of those security sensitive parts of the VMMs and vTPM manager which are responsible for measuring the vRTM.
* Compromise of Boot Sequence
  + A compromise of the boot sequence is detected by measurements made into the pTPM PCRs, satisfied by following the PC Client platform specification
* Compromise of SMM
  + A compromise of the SMM is detected by measurements made into the pTPM PCRs, satisfied by following the PC Client platform specification
* Compromise of VMM kernel
  + A compromise of the VMM kernel is detected by measurements made into the pTPM PCRs, satisfied by following the PC Client platform specification extended into the boot loader or augmented with DRTM
* Compromise of vTPM Manager
  + A compromise of the boot sequence is detected by measurements made into the pTPM PCRs, extended through the boot loader or via DRTM and in a trusted sequence also being required to measure the vTPM Manager.
* Compromise of the backup mechanism for vTPMs
  + The backup mechanism is dependent on the correct operation of two parts: the local backup operation and the remote backup server.  
    The primary countermeasures for the local backup operation are the same for any other component of the TCB: countermeasures to protect the correct boot of the system (todo: put in reference). Countermeasures to ensure that secrets are only available to correctly booted systems tie the correct boot to the certification of the backed up content.
* Rollback of vTPM secrets
  + A rollback of vTPM secrets is detected via using the monotonic counter of the pTPM being incremented at each launch of the vTPM manager which is successful in recovering the vTPM secrets, and having the then current monotonic counter being saved with those secrets.  Comparison between the two values should always differ by 1, thus proving a rollback attack has not happened.
* Loss of vTPM user privacy
  + Since vTPMs are \*not\* immutable, a re-instituted vTPM will resolve loss of privacy of a user caused by the unlikely event of user tracking via EK.
* Integrity (Change) of vTPM secrets and vPCRs
  + Only a verified vTPM is allowed access to vTPM secrets, which must be stored with both confidentiality and integrity attributes in their encryption (either via crypto mode or via encrypting the hash of the data along with the data).
* Confidentiality of vTPM secrets
  + Only a verified vTPM is allowed access to vTPM secrets, which must be stored with both confidentiality and integrity attributes in their encryption (either via crypto mode or via encrypting the hash of the data along with the data).  While in use, the VMM is responsible for maintaining the isolation necessary to prevent leakage of vTPM secrets from the vTPM.
* Attack by a hostile VM running under the same VMM
  + It is the VMM’s responsibility to provide isolation preventing attack from a hostile VM.
* DMA Attacks against VMM and VMs
  + It is the VMM’s responsibility to provide isolation preventing attack from a hostile VM. It is noted that DRTM provides protection utilizing either IOMMU or VT-d.
* Attacks on associations between VMs that share a vPlatform
  + A measured VMM and vTPM manager are responsible for maintaining the associations between VMs that share a vPlatform.
* Attacks leading to multiple instantiations of a vTPM
  + A measured VMM and vTPM manager are responsible for maintaining the associations between VMs that share a vPlatform.  Cryptographic protections are used to maintain the association of a VM platform with its vTPM, such as maintaining a copy of the hash of the VM with the vTPM secrets in an encrypted blob when a VM is closed.
* Attacks leading to poor or no entropy seeding the generation of keys, etc.
  + VMM entropy should be fed (at least in part) from a pTPM RNG. If this attack vector is of particular concern, the vTPM Factory may use a high-quality entropy source to generate the vTPM RNG seeds.
* Attacks on backed up, migrated or quiesced VM and VMMs
  + Rollback attacks are previously mentioned.  vTPM secrets are previously mentioned. VM secrets can be encrypted using vTPM protected secrets, along with integrity measurements based on vTPM PCR contents or integrity-protected reference values checked by trusted components.  This does not prevent Denial Of Service attacks, but one reason backup solutions are important is to mitigate DoS attacks.
* Attacks against integrity of attestation reports
  + Standard nonce protocols are used to prevent replay of attestation reports; signatures over the attestation reports prevent counterfeiting; certificates of the keys used to sign the attestation reports provide proof of the integrity of the key.
* Attacks against certificates used by the VMM, vPlatform, vPlatform Manufacturing Authority, etc.[I1]
  + Certificates used by the VMM,vPlatform, etc. are signed by a pTPM key, locked to PCRs which represent those VMMs.  The pTPM key itself is signed by an AIK, which is attested to with the EK, which is attested to by the chip OEM.
* Migration to an Attacker
  + Migration of secrets can only be accomplished by a vTPM with access to those secrets, which has to be in an approved state to unseal those secrets.  An approved vTPM follows policy.  Policy prevents migrating to an attacker.

# Hardware Platform Support

## Initial pTPM provisioning goals (informative)

A Trusted Virtual Platform runs on a physical Trusted Platform (pPlatform). The pTPM must be configured to enable trusted attestation of virtual machines, extending the chain of trust from the physical environment to the virtual environment, as well as to support the secure storage of vTPM secrets. To do so, the pTPM must be provisioned with appropriate keys and credentials. Provisioning of the pTPM may also include such actions as initializing a monotonic counter to support an anti-rollback feature for virtual platforms (see 11.2), inserting public keys into NVRAM for verifying enterprised-approved software, or storage of DRTM or UEFI launch policies.

## pTPM Configuration Requirements (normative)

After provisioning, the following pTPM resources must be properly configured:

### pTPM state

The pTPM MUST be turned on (activated and enabled) and have an owner.

### pTPM keys and certificates

The pTPM SHALL have the following keys:

* An AIK (in TPM 2.0, non-migratable restricted signing key), which MUST be used to certify the VFK and VPCK using the TPM\_CertifyKey operation. The AIK MUST be certified by an external entity (e.g. CA); this external entity SHOULD be trusted by any appraisers which are expected to evaluate the trustworthiness of the platform and its VMs.
* The EK, which SHOULD be certified as described in the TPM specifications, to allow the platform to be remotely re-provisioned in the event of hardware failure or misconfiguration.
* The SRK, which SHOULD either be a no-authorization key or whose user authorization SHOULD be the well-known secret.

The pTPM MAY additionally have the following keys:

* A Storage key, which we shall refer to as the vPlatform Manager Data Key, or VPMDK, used to seal vPlatform Manager Data. (If this key does not exist, the SRK or another Storage key without the following constraints may be used; however, in this case, every sealed vPlatform Manager Data blob SHALL have PCR constraints as described in the second bullet.)
  + The VPMDK MUST be usable only by the vPlatform Manager; this constraint can be imposed using locality, authorization values, limited access to the TPM driver, or other mechanism.
  + The VPMDK SHOULD be PCR-constrained; if it is, the PCRs SHALL contain measurements of the VMM, vPlatform Manager, and any other components relied on for the trustworthy execution of the vPlatform Manager.
  + The VPMDK MAY be replaced over the course of the platform's lifetime; in all cases, the VPMDK SHALL continue to be PCR-constrained as described above if it was originally, even if the values change.
* A Binding key which an IT Authority can use to send securely encrypted software or policy updates to the system. In this case, the public portion SHOULD be sent to the external authority. The binding key MAY require authorization values or PCR constraints to use locally if additional security is desired; for example, an update of the authority-approved vTPM software measurements may wish to be constrained to the vTPM Manager. This key SHOULD be certified using the AIK above.
* An AIK which is used to certify eAIKs (see Section 10.6.2.5), if used by the system. This AIK SHOULD be constrained to only be usable by the vPlatform Manager using either PCR constraints, locality, or authorization values. An external appraiser SHOULD be able to verify the constraints on the key in order to establish trust in the vPlatform Manager. It MAY be the same key as the AIK listed previously.

Either the following keys SHALL be created in the pTPM; or there SHALL be a CA capable of issuing equivalent certificates:

* A non-migratable SHA1, DER, or SHA256 (TPM 2.0 only) signing key, which we shall refer to vTPM Factory Key, or VFK.
  + The VFK MUST only be usable by the vTPM Factory; this constraint can be imposed using locality, authorization values, limited access to the TPM driver, or other mechanism.
  + The VFK SHOULD be PCR-constrained; the pTPM PCRs SHALL contain measurements of the VMM, vPlatform Manager, vTPM Factory, and any other components relied on for the trustworthy execution of the vTPM Factory.
  + The VFK MAY be replaced over the course of the platform's lifetime; in all cases, the VFK SHALL continue to be PCR-constrained as described above if it was originally, even if the values change.
  + The VFK SHALL NOT be trusted to sign TPM internal data, such as quotes or CertifyKey certificates.
* A non-migratable SHA1, DER, or SHA256 (TPM 2.0 only) signing key, which we shall refer to Virtual Platform Certification Key, or VPCK.
  + The VPCK MUST only be usable by the vPlatform Manager; this constraint can be imposed using locality, authorization values, limited access to the TPM driver, or other mechanism.
  + The VPCK SHOULD be PCR-constrained; the pTPM PCRs SHALL contain measurements of the VMM and vPlatform Manager, and any other components relied on for the trustworthy execution of the vPlatform Manager.
  + The VPCK MAY be replaced over the course of the platform's lifetime; in all cases, the VPCK SHALL continue to be PCR-constrained as described above if it was originally, even if the values change.
  + The VPCK SHALL NOT be trusted to sign TPM internal data, such as quotes or CertifyKey certificates.

### pTPM Certification

The pTPM SHALL possess one or more of the following:

* An Endorsement Credential certifying its EK
* An Attestation Identity Credential certifying an AIK belonging to this pTPM
* A certificate in an enterprise-preferred format, certifying the EK or AIK

The credential(s) SHALL be usable in combination with the pTPM to certify other pTPM keys in a way that appraisers of the platform can verify. In a 1.2 pTPM, this SHALL involve the use of the TPM\_ActivateIdentity and/or TPM\_CertifyKey (TPM 1.2) or TPM2\_Certify (TPM 2.0) commands; Signing and Legacy keys SHALL NOT be used to certify other TPM keys.

The credential(s) SHOULD be signed by an authority which all appraisers of the platform can trust.

<informative comment> It is critical that the pTPM which supports a Virtualized Trusted Platform is certified in such a way that external parties wishing to establish trust in the system can do so. Without trust in the pTPM, an appraiser cannot trust the state of the system or the security of any keys or data stored on it. The credentials listed here offer distinct ways of establishing that trust. With a credential for an EK and appropriate CAs (PCA or ACA), we can acquire credentials for an AIK; with a credential for an AIK, we can use the TPM’s built-in certification functionality to establish trust in other TPM keys, such as the VFK or VPCK.</informative comment>

## Non-pTPM Provisioning Requirements

### Boot sequence

The platform MUST follow the TCG PC Client specifications for setting boot sequence. The BIOS MUST be set to a correct boot order (selected by the IT organization).

<Informative comment> One example would be to only allow the system to boot from the hard disk, so as to make offline examples more difficult. Another would be to use a drive which only allows access to stored data if the system has booted correctly.</informative comment>

### Security Launch Policies (normative)

If security launch policies are being used to secure the physical platform boot, the system SHALL be configured to ensure that any security launch policy updates can only be performed by components which are part of the Trusted Computing Base and measured into the pTPM on each launch. The security launch policy SHOULD be remotely verifiable.

### IT Authority Keys (normative)

The platform MAY have public keys stored on them for integration with the IT authority. These MAY include the following, or others as needed:

* Public keys used to encrypt secrets to send to a backup server
* Root public keys for certificate authorities used by the system
* Enterprise signing keys used to verify authorized updates

In all cases, if such keys are stored on the system, they SHALL be protected from unauthorized changes.

### Protecting IT Authority Keys (informative)

In order to keep these public keys safe from change, all that is needed is proof that they have not changed. One way to gain this proof can be obtained by storing the hash of those values in an NVRAM location set up in the TPM with the following characteristics:

* It is generally readable
* It can only be written by an Administrator
  + The NVRAM can be set up during the IT provisioning process
  + The NVRAM is set to be TPM\_NV\_PER\_AUTHWRITE, with the appropriate authorization.
  + The NVRAM will then only be updatable by entities with the IT-created authorization.
* No two TPMs should have the same ownerAuth.

It is likely that none of these values will ever have to be revisited.

### Procedure for Initial pTPM Provisioning (informative)

1. Take ownership – set ownership to a well known secret (20 bytes of 0s).
2. Store CA public key (CAPK) on client system, if applicable
3. Store Backup public key (BPK) on client, if applicable
4. Create Rollback Monotonic Counter (see 11), if using.
5. Generate Attestation Identity Key (AIK), VFK, VPCK, VPMDK (if using)
6. Certify pTPM keys, including VFK and VPCK.

* Any mechanism for certifying these keys will work. For example, the mechanisms described in the TPM specification, with a Privacy CA certifying the AIK and the AIK certifying all other keys. Alternately, a enterprise CA could certify each TPM key directly based on some trusted provisioning process having occurred.
* Regardless of the mechanism used, the credential for the VFK must include sufficient information for a remote appraiser to establish trust in the vTPM Factory.
* All certificates should include the machine’s serial number or other unique identifier, unless this is a privacy sensitive system, in which case they should not contain the identifier.
* Certificates should be stored locally on the platform, so they can be provided as part of remote authentication or attestation transactions. Optionally, a Certificate Authority may store a copy of the certificates in addition or instead.

1. Optional: Creates a DRTM or UEFI launch policy
2. Optional: Store DRTM or UEFI launch policy in NVRAM location
3. Reboot platform to get platform SRK values in correct state to be measured.

# Software Architecture Components

All components described in this section are abstract, allowing us to define system requirements without restricting compliant implementations to a single architectural approach. An implementation of this specification MAY have a single real component implementing the roles of multiple abstract components, or split an abstract component into multiple real components, as opposed to implementing the system with the exact set of distinct components listed here However, all requirements listed MUST be met by the appropriate real components, and when determining whether a system meets this specification each abstract component MUST be associated with one or more real components for verification.

## Virtual Machine Manager (VMM)

### VMM Requirements (normative)

* The VMM SHALL provide isolation of vTPMs, so that information does not flow in or out of vTPMs except via approved interfaces. [See the Threat Model section for discussion of threats and countermeasures.] The VMM SHALL provide isolation of all components, such as the vPlatform Manager, which protect vTPM data or encryption keys. The VMM SHOULD provide isolation between all Virtual Platforms and VMs except for interfaces described in the vPlatform Credential, such as VM introspection or shared network resources.
* The VMM SHALL provide a mechanism or mechanisms allowing the vPlatform Manager to only release vTPM secrets to the appropriate vTPM during a load operation, and ensuring that vTPMs send their secrets only to the vPlatform Manager during a save operation. This mechanism should ensure not only that only the desired recipients can read the data, but that no unauthorized person can modify the data in transit.
* The VMM SHALL provide a mechanism or mechanisms allowing each vTPM to identify the vRTM in order to support a secure boot chain of trust. The VMM SHALL ensure that communications between the vTPM and the vRTM are both secrecy and integrity protected.
* If DRTM is in use, the VMM SHALL be measured iand launched using the DRTM. If DRTM is not in use, the VMM SHALL be measured and launched as part of the SRTM chain of trust. The VMM measurements SHALL be stored in the pTPM. Any executables or data, including security policies, that are relevant to the VMM’s meeting the specified security properties or behaviors SHALL be included in these measurements.
* The VMM SHALL provide a mechanism by which VMs and vTPMs can be measured before or during launch. This mechanism SHALL support the placement of vTPM measurements into pTPM PCRs, although it MAY store measurements internally and place the measurements into resettable pTPM PCRs when needed. The mechanism MAY support the placement of other VM measurements into either vTPM or pTPM PCRs, or both, as needed to ensure a solid chain of trust from the physical RTM through the vTPM PCR contents. The mechanism MAY support using these measurements outside of PCRs; for example, in reports requestable by an appraiser.
* When measuring VMs, the VMM SHALL at a minimum measure the VM executables and configuration files; it MAY additionally provide a mechanism to measure additional data.
* The VMM SHALL additionally measure or provide mechanisms to measure any components of the TCB, such as the vPlatform Manager, which are not already measured by the DRTM. Measurements of these components SHALL be placed in the pTPM PCRs; additional measurements of these components MAY be stored and used in other fashions, such as in reports.
* The VMM MAY provide a mechanism by which the vTPM can associate localities with specific VMs or other system components. If the vTPM’s vPlatform contains multiple components expected to make use of the vTPM, the VMM SHOULD provide this mechanism. If virtualized localities are supported, the VMM SHALL provide a mechanism by which the vTPM can reliably identify the components. The VMM SHALL provide a mechanism by which locality-identified components can communicate with the vTPM in a secrecy and integrity protected fashion. The vTPM MAY assign localities to VMs and other system resources outside of the Virtual Platform.
* The VMM SHALL perform the operations required of it in the Platform Operation section of this specification.
* The VMM SHOULD be isolated from possible threat vectors, such as the network or untrusted software, to the maximum extent possible.

### VMM Description (informative)

The VMM’s role in a trusted Virtualized Platform is primarily about isolating and identifying VMs. This role is critical; if the VMM fails to protect a vTPM from another VM, the resulting leak of secrets could result in a trusted vTPM becoming completely untrustworthy in an invisible and undetectable way. If the VMM allows an untrusted VM to identify itself falsely to the vPlatform Manager as a vTPM, it might allow the untrusted VM to steal the vTPM’s entire data. Even momentary failures of the VMM can be catastrophic. In other words, we are relying on the VMM’s security guarantees to provide many of the security guarantees for vTPMs that hardware provides us for the TPM.

The VMM’s security properties which support the security property guarantees in certificates must be measured into the pTPM.  In the event that the VMM is booted without the security properties necessary to maintain the security guarantees of the vTPM, the vTPM secrets must not be released to the VMM.  The design relies upon PCR values of the pTPM to provide this protection, whether enforced locally using sealed data or remotely via attestation.  As a result it is necessary that these VMM security properties be measured into pTPM PCR values which can then be used for this protection. This may be done with a combination of SRTM and DRTM trust chains.

The VMM must maintain isolation of the vTPM subsystems to mimic the hard boundary provided by the physical case of a physical TPM.  In particular, protected areas of the TPM must not be readable or changeable by any other VMs except through standard published TPM interfaces. Some attacks that are of particular importance for the VMM to defend against include:

* DMA attacks
* Software breaking out of a VM to the VMM
* Software breaking out of one VM into another VM
* Software breaking out of a virtual Platform

Note that introspection of one VM by a helper VM is specifically allowed, as long as that interaction is part of the Virtual Platform’s platform definition; VM introspection is a valuable tool for detecting malware and performing runtime security checks, and this architecture is designed to support its use.

The VMM itself, due to the nature of a modern virtualized system, has access to all the secrets in a vTPM.  If it is compromised after it has booted, the security of the vTPM is then also compromised.  This type of compromise is typically not visible in pTPM PCRs, as PCRs normally contain only boot-time measurements, and at best provide information about vulnerable software versions.  As a result, all communications with the VMM after it has booted have to be carefully controlled, to minimize attack vectors.  Any changes to the VMM, such as a security policy update or patch, that might result in a change to the security property guarantees must be preceded by locking all secrets.  Renewed access to those secrets must be preceded by certification that the security property guarantees are in place.  Typically this is done by sealing data to approved pTPM PCR values (which, in the case of boot-time measurements, would require a reboot of the machine to verify), or attesting to approved pTPM PCR values preceding a release of keys necessary to decrypt that data.

Some recommendations to minimize the attack surface include:

* Direct access from the internet to the VMM should not be possible
* Access to the VMM from a separate management network is allowed, but care must be taken to maintain strict isolation between the management network interface and any other interface
* All direct access to the VMM should be authenticated, and use secure channels
* It is recommended that network device drivers be maintained in separate VMs that are isolated from the VMM

Machines into Virtual Platforms (see Section 8.5). The VMM is also responsible for enforcing appropriate access control constraints both within and between Virtual Platforms.

## vPlatform Manager

### vPlatform Manager Requirements (normative)

* The vPlatform Manager SHALL perform the operations required of it in the Platform Operation section of this specification.
* The vPlatform Mananager SHALL create the Virtual Platform, as described in Section 9.2, and ensure on each boot that the Platform meets the requirements of Section 8.5.
* The vPlatform Manager SHALL create the Virtual Platform credentials, as described in Section 10.4.
* The vPlatform Manager SHALL securely store VDKs and hashes of the most recent vTPM data, to allow verification of data recency and integrity by the vTPM. The vPlatform Manager SHALL ensure that rollback protection mechanisms are in place to prevent old vTPM data from being loaded, including as a result of a rollback attack using the Manager’s old data.
* The vPlatform Manager SHALL encrypt its internal secrets, including all VDKs, before storing them; keep its stored data up to date to prevent rollback attacks, and take steps to protect the integrity and secrecy of its stored data, including all authorization values or special permissions that grant it access to its exclusive TPM keys, from all entities. The encryption SHALL use the pTPM, or SHALL use a symmetric key which is itself sealed with a pTPM key.
* The vPlatform Manager SHOULD NOT be directly connected to the network. It SHOULD be isolated from external requests and untrusted components.
* The vPlatform Manager SHALL be measured into the pTPM and be remotely appraisable.
* The vPlatform Manager MAY act as the exclusive system interface to the pTPM, in order to ensure that no unauthorized components make use of vPlatform Manager pTPM keys.
* The vPlatform Manager MAY provide support for deep attestation via mechanisms described in Section 10.6, such as creation and certification of eAIKs.

### vPlatform Manager Description (informative)

The vPlatform Manager is responsible for ensuring that vTPMs are only launched in association with the correct VP, and for protecting a given vTPM’s secrets from being loaded by the wrong vTPM. Although the vPlatform Manager does not have the power to prevent untrusted “vTPMs” from being launched by the VMM, the vPlatform Manager is responsible for only releasing trusted vTPM secrets if the VMM and other trusted components have correctly launched the vTPM and associated vPlatform. (See Section 9.4 for more details.)The vPlatform Manager may also act as an intermediary to the pTPM, in service of this role.

The vPlatform Manager, which might be a subcomponent of the VMM or a trusted VM, is responsible for reinstantiating a vTPM for each VM or Virtual Platform which requires a TPM. It may also be responsible for calling the vTPM Factory to create a new vTPM as needed; this is one reason many impementations may choose to combine the vTPM Factory with the vPlatform Manager. The vPlatform Manager is responsible for providing the vTPM with its encrypted data and verifying the integrity of the data, as well as tracking the long-term association of vTPMs and Virtual Platforms. The vPlatform Manager, in cooperation with the vTPM, is responsible for securing the data of the vTPM against data at rest attacks; it may also support backup and restore of vTPM data using a public backup server key.

The vPlatform Manager may also be responsible for populating the SRTM vPCRs of the vTPM with appropriate values, possibly with the help of the vBIOS.

Because we rely on the vPlatform Manager’s good behavior to protect secret data such as VDKs, to correctly associate vTPMs with the correct platforms, and in some cases even to provide measurements to the vTPM or pTPM, it is essential that the vPlatform Manager’s operation be correct. A corrupt vPlatform Manager could result in vTPM PCR values being unreliable even in the most minimally trusting architectures; in most architectures, a corrupt Manager could prevent the proper update of vTPM data, could allow rollback attacks, or could even cause all vTPM secrets to be leaked. This makes proper protection of the vPlatform Manager extremely appropriate, and just as with the VMM, precautions should be taken to minimize exposure of the Manager. For example, we recommend that there be no direct network interface to the vPlatform Manager; if network launch of virtual platforms is desired, an intermediary component which itself has a limited API to communicate with the Manager and is treated as potentially untrustworthy would reduce the direct risk of Manager corruption.

In addition, because so much of the way we assure the vPlatform Manager’s good behavior is based on pTPM PCR state and access to particular resources such as the VPMDK, it is essential that only the Manager be capable of using those keys if a good Manager is running on the system. Otherwise, an adversary could use corrupt software running on the same system as a good Manager and act in the Manager’s role, resulting in the same corrupt Manager problems we describe above. If a system allows the pTPM localities to be configured in such a way that only the Manager and other TCB components have access to decrypted Manager-specific resources, this is the most efficient choice and allows for other components to use the pTPM as needed. However, if locality constraints are infeasible in a given system, then it is highly recommended that the Manager serve as gatekeeper to the pTPM, and only allow transactions from other components which would not compromise the Manager or its secrets.

## vTPM Factory

### vTPM Factory Requirements (normative)

* The vTPM Factory SHALL be responsible for the initial creation of vTPMs, including providing them with their factory default settings.
* The vTPM Factory SHALL certify the EK of each vTPM it creates (issuing the Virtual Endorsement Credential, Section 10.2). The vTPM Factory MAY certify additional vTPM keys.
* The vTPM Factory SHALL perform an initial encryption of the vTPM data before the vTPM is first launched, to ensure that the constraints certified in the vTPM EC are met. (See Section 9.2.3 for details.)
* If migration is allowed, the vTPM Factory MAY exist on a separate physical platform from the ones on which created vTPMs will run.
* The VFK or CA key used to sign vTPM credentials SHALL be protected from unauthorized use. The key SHOULD be protected using a pTPM imposing PCR or locality constraints; it MAY be protected via other mechanisms which offer similar or better protection against unauthorized use.

### vTPM Factory Description (informative)

The vTPM Factory is responsible for creating and issuing initial certificates for vTPMs, just as a manufacturer is responsible for creating and issuing initial certificates for pTPMs. Although all components listed here are abstractions, the vTPM Factory in particular we expect to often be implemented as a subset of the vPlatform Manager; however, the Factory may be a separate component or even located off-platform. (For example, in some high-security use cases, there may be interest in creating vTPMs and their initial random seed using special high-randomness hardware; or if verifying pTPM-rooted credentials is difficult in a particular infrastructure, a Factory might use a software CA key to create and certify all vTPMs in an enterprise. However, if pTPM-rooted credentials are not possible, it is important that an enterprise have some mechanism by which the key is protected, such as the use of special cryptographic hardware, isolated machines with protected vaults or guards, etc.)

The vTPM Factory has two primary trust roles: issuing the vTPM credentials so that other entities can verify that the vTPM is genuine and operates only in accordance with certain guarantees, and encrypting the vTPM data so that there is a cryptographic assurance that the guarantees listed in the credentials must be met for the vTPM to launch. Although we do not specify what guarantees are required, since it will vary by system, example guarantees we expect to see include:

* “vTPM only bootable in presence of specific pTPM PCR values”: a set of PCR values that will be used to seal vTPM data, including measurements of the TCB and the vTPM itself, normally stored in a resettable pTPM PCR by the vPlatform Manager
* “vTPM only bootable in presence of specific pTPM PCR values approved by a third party, identified by public key”: the third party is responsible for updating the approved pTPM PCR values as described In the previous example. This is more update-friendly, but requires that the appraiser trust the third party to correctly enforce a system update policy.
* “approved by a third party identified by public key”: the third party would be provide the vTPM data decryption key on vTPM launch, usually in response to an pTPM quote or other attestation. Note that this allows an expert party to evaluate machine state based on an attestation, allowing easier updates to software without the fragility of PCR-sealed data, but at the cost of requiring network access to the third party whenever vTPMs launch. This solution also effectively outsources the decision of whether to trust the vTPM to the third party unless deep attestation (10.6) is performed; this may be considered a feature in enterprise environments..

## vTPM

### vTPM Requirements (normative)

* The vTPM SHALL implement a 1.2 or later TPM specification. The shutdown and restart of a Virtual Platform shall be treated as the equivalent of a shutdown and restart of a physical platform.
* The vTPM’s internal secrets and the integrity of its data SHALL be protected whenever the data is not in the vTPM-controlled memory space, by encrypting vTPM data before it is stored, and by saving its data with appropriate rollback prevention mechanisms whenever its non-volatile state changes. (See Sections 9.3, 9.6, 11)
* vTPMs SHOULD maintain a distinction between software and data, and use a standardized serializable format to store the data, in order to simplify software upgrade and migration.
* In the event that vTPM software is changed, it SHALL NOT be usable by associated VMs until a reboot of the Virtual Platform causes it to be remeasured. Any cached keys which rely on PCR state, or any volatile data intended to last for a single session (such as TickNonces) MUST be cleared or reset as appropriate whenever vTPM software is changed.
* Long-term vTPM state, such as the Endorsement Key, monotonic counters, authorization values, etc, SHALL be maintained over reboot and upgrade of the vTPM; long-term state SHALL NOT change except via the mechanisms described in the TPM specification. If backups are supported, there SHALL be mechanisms in place for local and remote parties to detect rollback.
* Only a single copy of a given vTPM SHALL be instantiated at a given time. If hot-swappable backups are desired, the backups MUST be a mirror of the primary copy at all times.

### vTPM Discussion (informative)

The vTPM serves the same role in a virtual platform as a pTPM does in a physical platform: it provides cryptographic services, identifies the platform using cryptographic keys, and serves as a root of trust for reporting on the virtual platform state. In the simplest case, where a single VM is associated with the vTPM, the VM has the same relationship with the vTPM as a normal OS has with a pTPM; however, virtual TPMs also can be part of more complex virtual platforms where different components are associated with different localitiesl Unlike physical platforms, localities can even be assigned to components outside of the vPlatform, such as the vRTM or shared measurement helper VMs.

This flexibility allows the vTPM to provide cryptographic keys distinguishing different VMs, or to provide an appraiser with an accurate picture of where a given measurement stored in a vPCR. originated, in both simple and complex architectures. The vTPM PCRs are primarily used to store measurements of the VMs (and sometimes non-VM services, such as virtual BIOSes) associated with the virtual platform, allowing a chain of trust from the vTPM through other software in the same way that the pTPM can be used to build a chain of trust from hardware through higher-level software, including the vTPM itself.

## Virtual Platform (VP)

### Virtual Platform Requirements (normative)

* A Virtual Platform consists of a primary VM and one or more “helper” components providing services to that guest, including a vTPM.[[1]](#footnote-1) Components may be VMs, or may be other processes on the system which provide services exclusive to the Virtual Platform; for example, in an implementation in which each vTPM is implemented as a separate isolated thread in a single VM, or inside the VMM, the vTPM would still be associated with a particular Guest VM and be considered part of its Virtual Platform.
* The primary VM of a Virtual Platform SHALL be measured into vPCRs 0-23 according to the PC Client specification. VTPM localities 1 and 2 SHALL be assigned to the primary VM. vTPM localities 3 and 4 MAY be assigned to the primary VM. vTPM locality 0 SHALL be unconstrained within the Virtual Platform (i.e., usable by any VM in the Virtual Platform). vTPM locality 0 MAY additionally be used by trusted components, such as the VMM or Domain Builder, which are not part of the Virtual Platform; all such components MUST be measured into the pTPM.
* Components in a Virtual Platform MAY be associated with localities in the vTPM. Localities 0-4 SHALL be associated with the primary VM, or (in the case of vDRTM) the component whose functionality is equivalent to the physical platform version. If the vTPM supports higher localities [15], other components (such as additional VMs) may be associated with localities above 4.
* The association between vPlatform components and locality SHALL be listed in the Platform Credential.
* The mechanisms by which locality-associated components are identified, and communications between those components and the vTPM are authenticated, SHALL be remotely verifiable
* Each VM SHALL only be a component of a single Virtual Platform. The vTPM associated with a Virtual Platform SHALL only boot in association with that Virtual Platform. Other VMs associated with Virtual Platforms SHOULD only boot in association with their Virtual Platforms. Virtual Platforms MAY boot without all VMs associated with the Virtual Platform; however, if it does, this SHALL be visible in the vTPM PCRs.
* A VM MAY provide services to Virtual Platforms which it is not a component of. Processes within such a service VM MAY be considered part of other Virtual Platforms. If a VM provides services to another Virtual Platform, it MUST either be measured into the vTPM of the served platform, or support a deep attestation mechanism that will allow a remote appraiser to verify both the measurements of the service VM and the relationship between the service VM and its served platform.
* All components within a Virtual Platform SHALL be measured into that platform’s vTPM PCRs, or have their measurements verified when the vTPM is allowed to retrieve its secrets. Measurements of the vTPM SHALL be verified when the vTPM is allowed to retrieve its secrets, and SHALL NOT simply be placed into the vTPM PCRs.
* The vTPM PCRs MAY contain measurements of components of other Virtual Platforms, if they are relevant to the secure operation of the vTPM’s Virtual Platform. If this is the case, those measurements SHALL be part of a remotely verifiable chain of trust; these components and the PCRs their measurements are stored in SHALL be listed in the Platform Credential. In the event of migration, these PCRs SHALL be invalidated; they MAY be reset and replaced by new measurements produced by a trusted component on the new system.
* If any Virtual Platform’s correct operation depends on the contents of another Virtual Platform, the other Virtual Platform’s state SHALL be verifiable via deep attestation.
* Virtual Platforms SHALL be isolated from each other except for defined communications interfaces allowed by the VMM. Interactions between VMs within a Virtual Platform SHOULD be isolated from each other except for defined communications interfaces allowed by the VMM. Communications interfaces are considered part of the Virtual Platform configuration, and SHOULD be reflected in a PCR (e.g., the configuration file the VMM uses to set up the Virtual Platform, or the security policy regulating access to shared resources, should be measured into a PCR, allowing a remote verifier to assess possible risks.).
* Each Virtual Platform with a vTPM SHALL have a Platform Credential describing the platform specification, as described in Section 10.4.

### Virtual Platform Discussion (informative)

Virtual Platforms (vPlatforms) are groups of VMs and other components which work in cooperation with each other in the same fashion that the various hardware components of a hardware platform do..

The simplest vPlatform would consist of a single VM, normally a general-purpose OS; however, the trusted vPlatforms addressed in this specification must additionally include a vTPM, to allow the platform to attest to its identity and state. Highly complex vPlatforms might include a user OS, several different virtual devices, measurement helper VMs, a vTPM, and a process running in a shared helper vPlatform which provides a secure interface to the user.. vPlatforms which do not contain a standard OS may exist; vPlatforms providing system services, for example, may wish to limit unnecessary features for ease of verification.

vPlatforms allow a system to take full advantage of the VMM’s separation of components, enabling security-sensitive functionality to be isolated while allowing the whole to operate as a single combined virtual system. Just as a physical platform generally contains an OS and a variety of hardware components, vPlatforms contain a primary VM (usually a user-accessible OS) and a variety of “helper” components, such as virtual devices. Note that the distinction between primary VM and helper components is only truly relevant when it comes to specific uses of the vTPM.:

* The PCRs used in the PC Cient specification to record the boot measurements of the platform will in a vPlatform contain the equivalent measurements of the primary VM. This provides an appraiser with reasonable backwards compatibility.
* The localities defined as controlled by the trusted OS and applications in the PC Client (localities 1 and 2) will be assigned to the primary VM.

“Helper components” are therefore not necessarily subservient to the primary; however, since he most common vPlatform is likely to have a user OS running in the primary VM and the helper VMs (if present) providing useful services to that OS, this terminology will be descriptive most of the time.

Although helper components are not required, isolating high-risk functionality can limit the impact a corrupted, malicious, or poorly programmed component can have; separating security-sensitive functionality can protect it from compromise of the primary VM. Features such as VM introspection, allow measurement-specialized helper VMs to assess the functionality of the primary VM or other helper VMs, isolating compromise detection from the components at risk.

Associating helpers components with a primary VM in a vPlatform allows the definition of reasonable expected behaviors for the entire group. Such behaviors might include limited information flow to or from other vPlatforms to minimize the risk of leaked data, or restricted access to hardware components to reduce side-channel vectors.

Normally, VMs in one vPlatform should not provide services to other vPlatforms, since the vPlatform model encourages isolation between unrelated components. However, in many systems, there will be a need for what we call helper vPlatforms, providing services to other vPlatforms. Often these will result from the need to limit access to shared hardware resources: for example, a system with a single network card might want to limit direct access to a single helper vPlatform acting as a virtual network switch. Other vPlatforms would connect to that helper vPlatform, which would be responsible for making sure that information did not leak between vPlatforms and access to the shared resource was distributed reasonably.

In some cases, a single VM may provide services to a number of vPlatforms, but each vPlatform would be using a single, isolated instance of the service: for example, some systems implement vTPMs as processes running in the vPlatform Manager’s VM. In these cases, it is often best to draw the vPlatform boundaries at the process level rather than the VM level, since each process is a helper component for a single vPlatform. The VM running the processes would then be responsible for meeting the vPlatform isolation requirements.

Where the boundaries between Virtual Platforms should be placed is up to the system designers; there are no hard requirements. Generally, if a component—be it a process, an application, or a VM-- provides services to, or about, a particular VM, it is probably part of that VM’s vPlatform: it will be measured into and make use of the same vTPM, allowing an easily shared chain of trust. If a component provides services to components of many different vPlatforms, it probably belongs in a helper vPlatform, whose vTPM will be used to provide deep attestations (10.6) to other vPlatforms as required to establish trust in the service. Choosing vPlatforms on a system is a tradeoff between the overhead of a complex system architecture vs. the advantages of isolation between components.

vPlatforms, like physical trusted platforms, must adhere to a consistent Platform Specifiation over time. Platform specifications define the contents and interpretation of vTPM PCRs, the association between localities and platform components, and other behaviors critical to establishing a chain of trust and performing an attestation of the platform. If these trust-critical properties changed over time, it would be impossible to establish full trust in any vPlatform component beyond the vTPM. As such, it is critical that vPlatforms be certified properly (See 10.4), particularly because vPlatforms are far more at risk of change (with software updates or malicious activity, for example) than physical platforms are. Should it be necessary to change the vPlatform Specification, it will be necessary to also create a new vTPM, since this is effectively creating an entirely new vPlatform.

## Migration Components

### Migration Component Requirements (normative)

* If the platform supports the external backup, restoration, and/or migration of vTPMs, the platform SHALL have a Migration Engine and at least one Migration Authority. These components MAY be the same if the Migration Authority is on-platform. The Migration Authority MAY be on a separate physical platform.
* The Migration Authority or Authorities SHALL be responsible for approving all actions which may result in vTPM secrets leaving the platform, including vTPM backup and restoration as well as vPlatform migration. Which Migration Authority is responsible for a given vTPM’s secrets SHALL be defined in the vPlatform Credential.
* The Migration Authority SHALL enforce the policy in the vEC (see 10.2) determining when vTPM data may leave the system and under what circumstances, as well as the policy determining when and where that data can be accessed.
* The Migration Engine SHALL be responsible for securely encrypting vTPM secrets for any action which may result in vTPM secrets leaving the platform. It SHALL ensure that a vTPM’s secrets only leave the platform with the approval of that vTPM’s Migration Authority.
* The Migration Engine SHALL be responsible for securely decrypting vTPM secrets and passing them to the vPlatform Manager along with all associated vPlatform data. The Migration Engine SHALL ensure that vTPM secrets are only decrypted if approved by that vTPM’s Migration Authority.
* The Migration Engine SHALL be in the TCB of the platform. It MUST be verifiable by an external appraiser.
* If the Migration Authority is on-platform, it SHALL be in the TCB of the platform, and MUST be verifiable by an external appraiser. If the Migration Authority is off-platform, it SHOULD be verifiable by an external appraiser.

### Migration Component Discussion (informative)

Migration Authorities and Migration Engines are optional components; because they are only used when a Virtual Platform is migrated to a new machine, or when a vTPM is backed up on an external machine, they are not required if the platform never sends vTPM secrets off of the platform. However, Migration Authorities and Engines are critical trusted components if backup or migration are supported.

Migration Authorities are responsible for determining to which external machines vTPM data should be sent, under what circumstances, and how that data should be protected when it leaves the system; they can be located on or off-platform. Generally, in circumstances where a central authority determines appropriate migration or backup policy for a large number of systems, an off-platform Authority is appropriate. In contrast, for systems where the Migratiion Authority will only be approving system backups or hardware replacements, an on-system Authority may be simplest.

Migration Engines are responsible for implementing the data protection policy enforced by the Authority, and for making sure that when a vTPM migration is attempted, the correct Migraiton Authority is consulted..

The system owner is responsible for setting the migration and backup policies that will be enforced by the Migraiton Authorities. Most systems will have one backup policy (for example, vTPM data will be encrypted with a trusted backup server’s public key and shipped to an offsite backup system every time the local vTPM’s non-volatile state changes) and one migration policy (for example, Virtual Platforms will only migrate when the physical platform is being replaced, with the owner approving both the migration and the destination system), pre-established by the owner. Each vTPM’s Endorsement Credential will indicate whether that vTPM is subject to the backup policy and/or migration policy (See 10.2.). Other systems, particularly servers where multiple parties are responsible for different Virtual Platforms, may have more complex policies which vary between Virtual Platforms.

Because compliance with vTPM security and behavioral requirements is dependent upon the structure of the trusted system, any time vTPM secrets leave the system, anyone relying on the vTPM must be capable of verifying the trustworthiness of any systems the secrets may move to, before the secrets migrate. It must be possible to evaluate what other parties may be able to access the vTPM data, under what circumstances, and what people or machines must approve in order for the vTPM to operate on another system. This means the migration and backup policies must be visible to appraisers seeking to decide whether to trust a given vTPM. If a vTPM is trustworthy on this system, but could be migrated at any time to an untrustworthy system, the vTPM itself cannot be treated as trustworthy.

In an enterprise context, the migration policy will usually identify the central enterprise authorities which handle backup and migration; for example, all vTPMs or Virtual Platforms might be backed up to an IT department server, and encrypted with an IT department key, to be decrypted only if the IT department determines that the original machine needs replacement. In home or small-business contexts, the policy would be more likely to require the platform owner’s approval, since migration would be rare and backup restoration more a matter of an individual’s needs rather than an IT department’s formal policy.

## vRTM (normative)

VMMs implementing trusted Virtual Platforms SHALL provide at least one of the two following options for initial measurement of the Virtual Platform’s primary VM:

1. **Emulated Boot Firmware Launches Primary VM**In this case, the VMM SHALL provide a virtual boot firmware environment for the VM launch similar to what a physical platform would provide. This environment SHALL include at a minimum vSRTM and vBIOS components; it MAY provide a vDRTM. It SHALL include any additional components necessary to extend a transitive trust chain from the vRTM or vRTMs through the primary VM’s launch. All RTM components SHALL extend their measurements into the vTPM. vPCRs SHALL be extended according to the pertinent PC Client specifications for boot measurements, assuming that the primary VM of the Virtual Platform is the booting PC Client system

In this scenario, the vRTM MUST implement one of the following options:

* 1. **vSRTM:**  
     For this case, the VMM and vPlatform MUST provide a vCRTM, and a SRTM compliant vBIOS. These components are executed as if they were on a physical platform and they result in the launch of the VM. The vTPMs PCRs are populated per the TCG PC Client Specifications.
  2. **vSRTM and vDRTM:**  
     If a VMM is capable of emulating dynamic launch instructions (eg. Intel, AMD, etc provide these in newer processors), then the dynamic launch of a VM can be implemented. Dynamic launch requires a vCRTM and a vSRTM enabled vBIOS so these MUST be provided in such an implementation. In addition, the implementation MUST include the additional virtual boot firmware components for a DRTM. Such implementations MUST comply with the TCG Dynamic Root of Trust for Measurement (D-RTM) standard [16].
  3. **Modified vSRTM and Dynamic Launch in the VM:**Some manufacturers have implemented proprietary dynamic launch schemes on physical platforms by modifying their CRTMs and SRTMs and doing the dynamic launch in the earliest portions of the operating system launch. Such implementations can be done for virtual platforms but are proprietary and outside of the scope of this specification.

If a security launch policy (eg. UEFI secure boot policy) is part of the emulated boot process for a Virtual Platform, it SHALL be protected against modification by unauthorized parties. It SHOULD be protected using the Virtual Platform’s vTPM, such as by storing in the vTPM NVRAM.

1. **No Emulated Boot Firmware – VMM or other Trusted Component Launches Primary VM**   
   Rather than running a vRTM, vBIOS, etc in the style of a physical device, a VMM or another component which has been measured into the pTPM and whose state is part of the vTPM’s chain of trust may configure and setup a VM container, place a VM into this VM container, and start it. In this case, the VMM is doing the VM environment/container setup that would have been done by a vRTM and vBIOS. This case has the following requirements:
   1. The VMM SHALL measure the code responsible for setting up the VM environment and place those measurements in the vTPM’s vPCRs.
   2. The Virtual Platform Credential associated with the VM being launched SHALL contain a definition of how the vPCRs are being populated by the VMM.
   3. The implementation MUST be able to present a full transitive trust chain to the VM for its continuation and use.

Regardless of which vRTM option is chosen, vRTMs SHALL NOT ever act as true **roots** of trust, where they are trusted implicitly (trusted without any evidence); the vRTMs SHALL be measured into the appropriate PCR not by itself (as is used in physical platforms) but rather by a trusted component, such as the VMM, existing outside of the Virtual Platform. Whichever component measures the vRTM SHALL be measured into the pTPM. It SHOULD be one of the components whose correct behavior is required for a vTPM to launch.

Either boot option MAY be used for non-primary VMs. These measurements SHALL be included in vTPM PCRs above 23. Which PCRs are used for which measurements SHALL be included in the Platform Credential.

# Platform Operation

## Booting and Measuring the System

### System Boot Context (informative)

The most critical part of the system boot process is ensuring that a chain of trust can be built from the hardware through the VM layer. This means that the boot process must measure each critical component in a remotely verifiable fashion, and ensure that the booted software is the same as what has been measured. Whenever possible, trust-critical components such as the VMM should be measured using the DRTM or a similarly reliable mechanism. The chain of trust should extend from the VMM up through the highest level of software or system policy relied on by our trusted VMs. (The chain will be extended into virtual platforms and trusted VMs when those are launched.)

### Requirements for System Boot (normative)

* The following components SHALL be launched before any Virtual Platform: the VMM, the vPlatform Manager, and any other components whose correct operation is required for trustworthy execution of vTPMs and vPlatforms.
* The VMM SHALL be booted with either the SRTM or the DRTM (and therefore measured into the pTPM). The VMM SHOULD be booted using the DRTM.
* The vPlatform Manager and all other TCB components SHALL be measured into the pTPM. These measurements SHALL be taken and extended into the pTPM by the RTM or by another component previously measured into the pTPM in a chain of trust from the RTM.
* If the VMM uses policy or schema files to determine security relevant behavior (for example, Mandatory Access Control policies on VM communication), these files SHOULD be measured into the pTPM as they become effective both during boot and run time.

## Creating a New Virtual Platform

### Virtual Platform Creation Context (informative)

This section covers the creation of a new Virtual Platform from scratch. This normally occurs when a new virtual machine requiring a trusted infrastructure is being deployed for the first time.

Normally, when a new vPlatform is created, a new vTPM will be initialized. The vTPM Factory, acting as the vTPM manufacturer, creates the vTPM’s initial data and issues a vEC (10.2). The vTPM’s new data are tied to the pTPM state and protected as described in Section 9.3. The vPlatform Manager, acting as the OEM, associates the vTPM with the new vPlatform, configures its internal data to ensure that the vPlatform and vTPM are always associated when the vTPM is saved or loaded,, and issues an appropriate vPlatform Credential (10.4). This ensures that vTPM PCRs maintain a consistent interpretation over the life of the vTPM.

The timing described here is not strictly required. In systems where a dedicated vTPM Factory exists off-platform, vTPMs may be created in bulk in advance of their association with a particular vPlatform. In these cases, the original VDKs created by the vTPM Factory would be decryptable by the approved vPlatform Manager rather than by the booting vTPM, to allow the Manager to correctly use a pTPM seal operation to associate the vTPM data and vPlatform. The issuance of the vEC, similarly, can be delayed until after the vTPM has booted and been provisioned, if there is a mechanism by which the Factory can verify the legitimacy of the vTPM and its vEK.

The diagram in 9.3.1 describes two possible techniques for creating a vTPM, depending on which approach to protecting vTPM data is used. In both cases, the intial vTPM data is the same; however, the mechanism for storing the symmetric key which encrypts the vTPM secrets differs.

We will follow these two techniques through the operation section, as example implementation options.

Although our diagrams and examples will generally assume that a vPlatform has a single vTPM, this is not strictly necessary. In particular, some systems may wish to have both a 1.2 and 2.0 vTPM associated with the same vPlatform. In this case, all requirements for each vTPM hold true; any measurements that would be put into vTPM PCRs during the vPlatform boot process should be placed into both, etc.

### Requirements for Virtual Platforms (normative)

* vPlatforms SHALL possess a vPC (see 10.4)
* The requirements listed in the vPC SHALL always be enforced on this vPlatform and its components, even if the vPlatform migrates.

### Requirements for Newly Initialized vTPMs (normative)

* A newly initiatialized vTPM SHALL meet the state requirements of a newly manufactured TPM after the platform has been assembled by the OEM, as described in [4], [17], including but not limited to the following:.
* A new vTPM SHALL contain all manufacturer-created values and default flag values,§ and MAY contain additional information such as initial credentials or keys.
* The Endorsement Key may be either created by the vTPM Factory and injected into the vTPM, or may be created by the vTPM itself. In either case, it SHALL be certified by the vTPM Factory or another authority. (See 8.3)
* New vTPM data SHALL be encrypted by a VDK generated by the vTPM Factory even if it has no secrets yet, in order to ensure that all guarantees of vTPM behavior provided by decryption of vTPM secrets (see 9.4) are met when the vTPM is first launched. The new vTPM SHOULD contain at least one secret key, usually the Endorsement Key, to allow identification of the vTPM for certification.

## Protecting vTPM Data

### vTPM Data Protection Context (informative)

Whenever a new vTPM is created, we must determine the technique we will use to protect the vTPM’s secrets. We describe later in this section four different techniques capable of meeting the vTPM data protection requirements listed below, although the use of other techniques is allowed. In all cases, we assume that the vTPM secret data is actually encrypted with a symmetric key (vTPM Data Key, or VDK) which is itself protected using one of the following mechanisms. This symmetric key allows us additional flexibility in protecting the data, and means we do not need to be concerned with the pTPM’s ability to encrypt the entirety of the vTPM data. New VDKs should be used, but are not required, on each save, in order to provide additional integrity protection and help guard against rollback attacks. Reusing VDKs provides additional attack opportunities to an adversary who manages to acquire a VDK at any point in time.

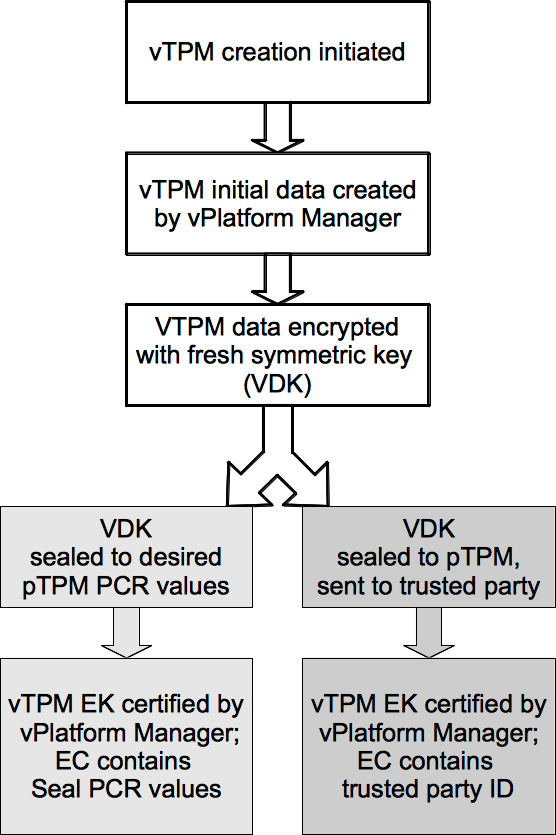


Figure : vTPM creation, showing two options for data protection. In light grey, vTPM data sealed to pTPM PCR values (9.3.3); in dark grey, third party approves release of vTPM data (9.3.4). We will be using these two colors to follow these two scenarios through the rest of the section.

In the above diagram, we show two of the main techniques we could use to protect vTPM data, as used when the vTPM is first created. On the left, we use PCR values to lock the vTPM to a particular state reflected in the pTPM. This approach is described in detail in 9.3.3. On the right, we use a generic pTPM key to protect our data, and send it to a remote third party, who will decide whether or not to release the key on a later boot of the vTPM. This approach is described in detail in 9.3.4. These approaches can also be combined, as described in 9.3.5.

These are not the only approaches to protecting vTPM data, and any approach that meets the requirements listed in 9.3.2 is within the guidelines of this specification. However, we expect most systems to use these techniques or variations on them, as they are simple, well-understood, and straightforward to implement.

### vTPM Data Protection Requirements (normative)

* vTPM secrets (i.e. private keys, the TPMproof nonce, read-constrained NVRAM data) SHALL be encrypted with a VDK when the vTPM is not in use, and SHALL be protected from unauthorized access while the vTPM is in use.
* Each vTPM SHALL use a unique, symmetric VDK. VDKs SHOULD be generated by the vTPM, VDKs SHOULD be freshly generated for each save, to minimize risk if old data is compromised.
* The VDK SHALL be encrypted as part of a storage chain of trust rooted in a hardware TPM when the vTPM is not in use (see 8.2.1), and SHALL be protected from unauthorized access while the vTPM is in use.
* Other vTPM data SHOULD be encrypted when not in use. Data here refers to information held within the vTPM, not the vTPM software itself.
* vTPM secrets MUST be constrained so they can only be decrypted when the vTPM and all software that the vTPM relies on for secure operation is running and in an acceptable state. What acceptable means MUST be defined in the vTPM credential.
* The vTPM data MUST only be available if the vTPM is running in association with its expected Virtual Platform.
* A hash of the most recent vTPM data SHALL be stored by the vPlatform Manager to allow the detection of rollback or modification attacks on stored, encrypted vTPM data.
* There MUST be integrity protection in place to prevent modification of vTPM data while not in use.

### Sealing vTPM secrets to PCRs (informative)

The simplest and highest-security, but most fragile, way of protecting vTPM secrets is to seal the VDK using a pTPM storage key and a set of PCR values which indicate an appropriate TCB the vTPM relies on. If, as in most systems, vTPMs are not measured directly into the pTPM when they boot due to highly variable order of operations (and so as to not leak information about which VMs are running), the vPlatform Manager can use a resettable PCR to extend the appropriate vTPM measurement into the pTPM for verification purposes. Because the vPlatform Manager is itself part of the TCB and thus measured and verified, this is a reasonable extension of our chain of trust. Other measurements critical to the secure operation of the vTPM, such as a Virtual Platform identifier, should be included as well, making sure that each such measurement is produced by a trusted component. The VDK (and thus the vTPM data, including all keys) is only retrievable when the correct pTPM PCR state is present. If the vTPM data has been created and sealed to an appropriate state once, and if a good vTPM will always ensure that its data is sealed to a good pTPM state again when the data is updated (either directly or via the VMM, whose measurement was also verified), then good vTPM data will always be accessible only to a vTPM meeting those same constraints. We will be following this scenario throughout this section using light grey diagrams as previously noted.

This approach has a few major advantages.

First, it requires no network-accessible verifier to be online in order to boot vTPMs in a trustworthy fashion, unlike the approach of 9.3.4.

Secondly, the vTPM state can be certified by a VMM or vPlatform Manager which creates or evaluates its initial sealed data, and the self-protecting structure will ensure that those same guarantees will be met for the lifetime of the vTPM and over arbitrary reboots. This allows long-term trust to be established in a vTPM root key (usually the vEK) and maintained for the life of the vTPM, in direct parallel to the mechanisms for certifying pTPM root keys. Trust in all other vTPM keys can be established from the vEK, as all vTPM keys will share the same constraints, and will only be accessible if the guarantees have been met. Normally, we would expect this vTPM Endorsement Credential to contain the sealed PCR values used in the vTPM’s creation, along with any update policies that might affect the values being used. This allows an appraiser to evaluate both the initial state of the system when the vTPM was firs created, and any changes (good or bad) that are likely to have occurred since.

And thirdly, a remote party can verify for themselves their trust in the entire infrastructure if the certification is done properly. The VMM or vPlatform Manager can sign the vEK Credential, including the pPCRs used to seal the VDK (and thus constrain access to the vTPM data), using a pTPM non-migratable signing key. The VFK is itself certified using the CertifyKey command, allowing the PCR constraints on it (which must, in this case, include the VMM and/or vPlatform Manager) to be verified. The verifier must decide if the pTPM AIK signing the CertifyKey credential is legitimate, and if so, if it accepts the various PCR constraints; but no blind trust is required. The verifier can decide for itself whether the platform state reflected in the PCRs corresponds to a trustworthy vPlatform Manager and VMM, and thus whether the vEK Credential is legitimate.

The disadvantage of this approach is that any changes to PCR values can cause Denial of Service problems for the vTPMs. While appropriate for some high-security scenarios, most real-world applications require more flexibility. One possible answer is to seal vTPM secrets, on each shutdown, to the latest measurements of “good” software, and for the vEK Credential to contain the system’s criteria for determining the measurements to use; in other words, to have a built-in update policy that includes changes to “good” PCR values as well as to software. Another is to use the hybrid approach described in 9.3.5, combining the local convenience of PCR bindings with the resiliency of a trusted appraiser’s approval

### Binding secrets to a key held by a third party (informative)

In this approach, the VDK is encrypted and then provided to a trusted third party. (Note that in cloud scenarios, this “trusted third party” might actually be the VM user’s home machine.) The encryption might be direct (such that the unencrypted key would be provided later by the third party) or in multiple layers (so that the VDK is sealed to a pTPM storage key with no PCR constraints, and the sealed blob provided to the third party). Our diagrams will assume the second approach, using the pTPM to protect the VDK.

When the vTPM boots, it must perform a Deep Attestation protocol (10.6) to convey information to a third party about the trustworthiness of the system and the vTPM. Such a protocol will generally use either measurements of the vTPM and the configuration of the vPlatform, extended into resettable pTPM PCRs, or an eAIK (See optional eAIK section, 10.6.2.4 and 10.6.2.510.6.2.510.6.2.5). If the state is acceptable, the trusted third party provides the vTPM with its VDK; the vTPM can now use it to decrypt the remainder of its data. We will be following this scenario throughout this section using dark grey diagrams as previously noted.

Here, the trust in the vTPM’s state is dependent on the third party; in this case, the vEK Credential must include the third party’s public key so that a verifier can evaluate the third party’s identity and key release policies. However, because the vTPM secrets will only be accessible when the third party has approved of the system, trust in vTPM keys can be established from the vEK just as in a pTPM.

We do not specify the exact protocol for how the third party communicates with the vTPM; however, it MUST provide a tight binding between the vTPM and the trusted third party (usually on the other end of a secure channel) who holds the encrypted VDK so that the VDK is kept secret. Similarly, we do not specify the exact protocol for providing the encrypted VDK to the appropriate third party, but it must guarantee that only the trusted third party can provide the VDK back to the vTPM, and that no copies are available locally or to untrusted parties.

This approach is highly recommended in cases where the VMM measurements are expected to change frequently, or where the trusted third party is itself the vTPM’s primary user. It is far more flexible and less likely to cause denials of service than the PCR-sealing approach of 9.3.3. However, it is also far more opaque to a remote verifier, as trust in the vTPM is entirely dependent on the third party; anyone trusting the vTPM must trust that the third party evaluated the VMM and vTPM software reliably both on this boot and on all previous boots, and also that the third party has kept the encrypted VDK as a protected secret at all times, so that no malicious party could have arranged for a bad boot of the vTPM at any point. This means that unless the trusted third party is itself a very well-secured system, this approach may be less trustworthy (if far more resilient) than 9.3.3.

### Hybrid Approach (informative)

In the hybrid approach, we wish to combine the advantages of the PCR-sealed and third party approaches described above. Each time the vTPM is saved, we encrypt the symmetric key twice: once sealed with a set of PCR constraints as in 9.3.3, and the other encrypted for a trusted party as in 9.3.4. The second encrypted blob is sent to the trusted party, which updates its most recent data for this vTPM; the sealed blob with PCR constraints is stored locally. On a normal vTPM boot, the PCR-constrained data is used, giving us all of the security guarantees we expect of a PCR-sealed boot process and eliminating the need for realtime network connectivity. (As discussed in section 9.6, we do not necessarily expect a vTPM to be saved every time it is used, so we can often benefit from not needing network access even when we expect to save data remotely on occasion.) However, in the event that the vTPM or its underlying software is updated or otherwise modified, breaking PCR constraints in an unexpected fashion, we can still retrieve our vTPM data by requesting access from our trusted third party, who will appraise the system and determine whether or not the data should be released.

The hybrid approach improves the recoverability and resilience of the fragile PCR-sealing approach, but means that we can take advantage of the speed and lack of network dependency of that same approach in the normal case, only using the third party’s approval as a fallback mechanism when the local PCR values change unexpectedly but for non-malicious reasons. The hybrid approach can also be used to provide resilience against a lost physical machine, in combination with externally backed up data. (See 9.8 for more extensive discussion of backup.).

When using the hybrid approach, the vEC must include both the PCR information and any update policies for the local release of data, and the public key for the trusted party, since an appraiser will need to trust both release policies in order to trust the vTPM..

### Binding secrets to an agent (informative)

This approach is similar to the trusted third party approach of 9.3.4, but in this case, the trusted party is on the same system as the vTPM, and its trustworthiness is guaranteed by pTPM PCR state. It takes advantage of the fact that it can be easier to measure simple components in a predictable fashion than complex ones.

In this approach, the vTPM secrets are bound to a pTPM key belonging to a software agent on the local platform, which in turn is bound to PCRs. Using this level of indirection, an agent whose security can be identified by PCR values is trusted to make the decision on the VMM and its configuration settings, perhaps based on signatures, and it then provides the key to the vPlatform Manager if it passes the agent’s assessment of trustworthiness.

Such an agent would have to be stable, so as to avoid problems with brittleness of the PCRs that represent its state, and would also have to not be susceptible to the ravages of a hacked VMM. Such an agent might be provided with these characteristics by launching it using DRTM before the launch of the VMM itself.

## Launching a Previously Executed vPlatform

### vTPM Launch Requirements (normative)

When launching a virtual platform that has previously been created, there are a few critical steps that must be taken.

* The vTPM’s secrets SHALL NOT be released unless the state of the vTPM and trust-critical software it depends on (such as the VMM or any shared libraries used by the vTPM) has been verified and approved; the secrets SHALL only be released to the vTPM itself.
* The Virtual Platform associated with the vTPM SHALL match the Platform Credential associated with the vTPM in order for the vTPM secrets to be released.
* A rollback mechanism SHALL be in place to verify that the vTPM data which is launched is the most recent saved state of the vTPM, and that the data has not been subject to rollback or modification attacks. (See Section 11 for one recommended mechanism.)
* Some entity (usually the vPlatform Manager) SHALL verify that this is the only copy of this particular vTPM running at this time.
* If the Virtual Platform associated with this vTPM is being restored from a suspended or frozen state, there SHALL be a mechanism to ensure that the state in the vTPM PCRs accurately reflects the current state of the Virtual Platform, and to halt launch if this is not the case.

### Approaches to Launching a vTPM (informative)

The diagrams below describe two different implementation approaches to launching a new vTPM, corresponding to the PCR-sealed protection mechanism described in 9.3.3 (light grey) and the third-party protection mechanism described in 9.3.4 (dark grey). No separate diagram is included for the hybrid approach described in 9.3.5, as launch in the hybrid approach is equivalent to one or the other of the listed approaches; or to both of them in sequence, where the remote approach is used if the local approach fails.

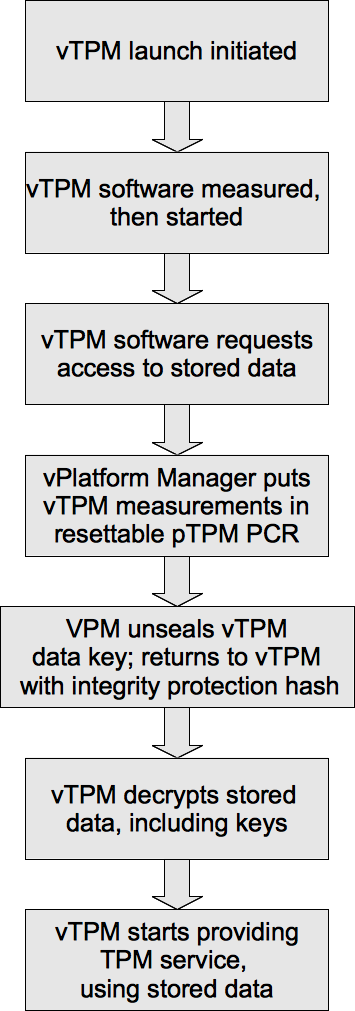
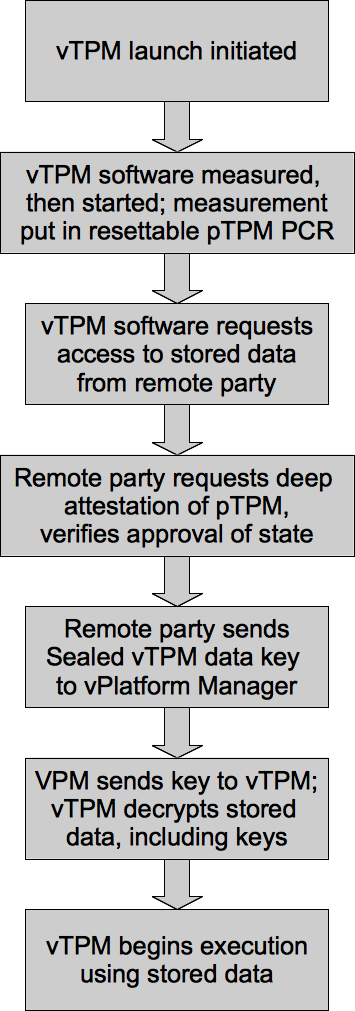
 

Figure : Left, launch of vTPM with PCR-sealed data. Right, launch of vTPM with remotely protected data, with pTPM-based attestation.

### Launching a vTPM using PCR-sealed data (informative)

* When an existing vTPM is launched-- even if it was created immediately beforehand-- the vTPM software is first measured.
* The vTPM measurements are provided to the vPlatform Manager, which extends them into a freshly reset resettable pTPM PCR. (They may also be stored internally, in order to reduce the risk of race conditions and to facilitate later deep attestation.) (A resettable PCR is used because a non-resettable PCR would require vTPMs to always be booted in the same order and with the same set active in order for the PCR values to be predictable and allow reasonable “good value” measurements.) The fact that the vTPM measurement is stored in a pTPM PCR will allow us to take advantage of the pTPM's data protection capabilities.
* An identifier of the Virtual Platform associated with the vTPM-- which might be as simple as an ID, or as complex as the combination of a name and the hash of a security policy-- is also extended into the resettable pTPM PCR, allowing us to verify the association between the vTPM and its VP.
* The vPlatform Manager uses the pTPM to unseal the VDK, which has previously been sealed to a set of expected pTPM PCR values. (See9.3.3). The operation will only succeed if the pTPM PCRs contain the expected measurements for the trusted computing base (including VMM and vPlatform Manager), vTPM, and VP.
* The decrypted VDK is sent to the vTPM, which retrieves its stored data (probably from the disk) and decrypts it. The vPlatform Manager additionally provides a previously stored integrity protection check, such as a hash of the encrypted data, with the vTPM uses to verify that its data is current and has not been modified,

Any use of the vTPM’s secrets (e.g. keys) thus proves that the pTPM PCRs were in the expected state at the time that the vTPM launched.

### Launching a vTPM using remotely protected data (informative)

The key challenge in launching a vTPM with remotely protected data is proving to the remote party that the vTPM is trustworthy via attestation. The diagram above uses a pTPM-based attestation technique, with vTPM measurements stored in pTPM PCRs and then quoted. This is only one way of providing data to the remote party. For other approaches, see the Attestation section (10).

The general structure of remote-protected vTPM launch is simple:

* The vTPM is launched and measured, usually by the VMM. It retrieves its encrypted data.
* The vTPM or vPlatform Manager contacts the remote party and requests the key for the vTPM’s data.
* The trusted remote party performs an attestation of the platform, including both the underlying TCB (VMM, vPlatform Manager, etc.) and the vTPM, as well as any required information about the associated Virtual Platform.
* If the remote party decides that the platform is trustworthy based on the attestation, it sends the sealed VDK back to the vPlatform Manager, or the unencrypted key back to the vTPM.
* If the VDK is sealed, the vPlatform Manager uses the pTPM to unseal it, thus proving that the correct hardware platform is in use, and provides the VDK back to the vTPM. (Sealing the VDK is optional; it provides additional protection for the VDK against compromise of the trusted third party when it is off-platform, but not sealing the VDK allows it to be used as part of the backup recovery mechanism in the event of machine loss or hardware failure.)
* The vTPM decrypts its data. Any use of the vTPM’s secrets proves that the remote party approved of the state of the vTPM and its underlying platform.

### vTPM PCR Contents on vPlatform Launch

* The vTPM’s PCRs MUST reflect the launch state of the vPlatform’s primary VM.
* If the Virtual Platform was resumed from a suspended state, the vTPM SHALL only restore its volatile state, including PCRs, if all VMs in the Virtual Platform, and any other components measured into vTPM PCRs, are in the same state that they were when the vTPM was suspended, and all other vTPM launch requirements are met. If there is an inconsistency in the state, the vTPM SHALL NOT be usable until the Virtual Platform reboots; the vTPM MAY NOT be usable until a platform owner or other authority verifies that the system is in an acceptable state.
* If the vPlatform contains multiple VMs besides the vTPM’s, each VM’s launch state SHOULD be reflected in vTPM PCRs.
* The Virtual Platform’s vRTM, whether static or dynamic, SHALL be measured and its measurements placed in the vTPM PCRs. These measurements SHALL be taken by a component which has been verified as part of the vTPM launch (e.g., by its measurements being included in pTPM PCRs that are checked when unsealing data).

<informative comment> The vTPM’s PCR contents reflect the state of the Virtual Platform in the same way that the pTPM’s contents reflect the state of the software running on the hardware platform. Whenever a vPlatform is rebooted, the values in the vTPM must reflect the new boot state of the VP. If the vPlatform is restored from a saved state, the values in the vTPM must be consistent with the current state of the VP; however, it is up to the vPlatform designers to decide what that means, depending on their needs. Critically, any trust decisions which an appraiser might make based on those PCR contents should be appropriate based on the current vPlatform state. A consistency check to ensure that the load state of the other vPlatform components is the same as the saved state would be one reasonable implementation option. Wharever choice is made, the information about how PCR contents will be changed or verified when the vPlatform is suspended and restored must be included in the platform specification.

One option that is explicitly NOT allowed is resetting non-resettable PCR values in such a way as to erase the previous contents, similar to a boot. This violates the expectations of good TPM behavior. </informative comment>

## Runtime Operation

### Runtime Operation Considerations (informative)

Because we have taken appropriate precautions when launching the vTPM to ensure that the platform is in an acceptable state, we can assume that while the VMM is running, it can be relied upon to protect the secrets of the vTPM, barring runtime compromise of the VMM itself. In particular, the VMM should:

* Prevent vTPM secrets from being accessed by other VMs, including other vTPMs.
* Prevent vTPM secrets being accessed through shared memory or other implementation mechanisms.
* Prevent cache manipulation attacks and other side channel attacks which might allow an attacker to learn information about a vTPM's keys.
* Prohibit access to the vTPM's standard interface by VMs that are not part of the vTPM's Virtual Platform, or by unauthorized VMs within the vTPM's VP.
* Prevent modification of the vTPM's runtime state by entities outside of the vTPM.

The VMM or vPlatform Manager may shut down the vTPM or revoke its credentials if there is some risk of violating these policies. A common policy in systems where TPM-based runtime monitoring is in use might include “the vTPM must shut down if the pTPM PCRs for the TCB change”; other policies might force shutdown or credential revocation in the event of software updates, security policy changes, or indications of possible malicious activity.

The primary purpose of these shutdowns is to ensure that a remote appraiser that establishes trust in a vTPM based on a certain set of behavioral assumptions has an opportunity to reestablish trust if there is a chance that those assumptions may no longer be true.

### Runtime Operation (normative)

* While the vTPM is running, the VMM MUST protect the vTPM's secrets and state.
* The VMM and other trusted domains MAY enforce constraints on the vTPM's operations. The VMM and other trusted domains MAY force a vTPM to shut down, lose access to certain secrets (see eAIKs, Section 10.6.2.4), have its certificates revoked in the local PKI, or otherwise enforce consequences if these constraints are violated.
* The vPlatform Credential (10.4) SHALL accurately reflect the vPlatform at all times.
  + Components required by the vPC SHALL be added to the vPlatform. Components allowed by the vPC MAY be added to the vPlatform.
  + If the vPC says a measurement of a given component will exist, the measurement SHALL be taken and placed in the corresponding vTPM PCR. If the vPC does not say that a measurement will exist, the measurement is not required.
  + Components not included in the vPC MUST NOT be added to the vPlatform.
  + vPlatform behavior SHALL correspond to the policies described in the vPC.
* If the vTPM is not running, the VMM MUST NOT release the vTPM's secrets.
* The system MUST ensure that the vTPM's PCRs accurately represent any associated VMs or Virtual Platform state in the event of a VM crash, pause, or restart.

## Saving vPlatform State

### State Saving Requirements (normative)

* Whenever the vTPM’s internal data is updated in a way that should be maintained over a boot, the vTPM SHALL save its state. There SHOULD be protection against race conditions that might result from a vTPM or vPlatform Manager crashing mid-save.
* When the vTPM’s state is saved, it SHALL do so in such a way as to protect its secrets and enforce any constraints guaranteed by its creation and credentials. This SHOULD include encrypting the vTPM’s state with a fresh VDK which is then encrypted; the fresh key, providing protection against data substitution attacks beyond the basic rollback protection mechanism.
* Whenever the vTPM’s state is saved, it SHALL provide the vPlatform Manager with a hash of its most recently saved data.
* The vPlatform Manager SHALL maintain a list of most recent data hashes for all vTPMs, to provide integrity protection of vTPM data against rollback and data modification attacks. This list SHALL itself have integrity protection against rollback and data modification attacks.
* The vPlatform Manager SHALL, upon updating its internal saved data, encrypt the data with a symmetric key known only to itself. This symmetric key SHALL be sealed using the VPMDK. The symmetric key SHOULD be freshly generated, to ensure secrecy. The encrypted data and sealed symmetric key SHALL be stored to disk.

### Saving State Examples (informative)

In the following diagrams, we describe two possible procedures for saving a vTPM’s state. In one procedure, the vTPM state is protected using the pTPM PCR values, as enforced by a sealed blob. In the other, the vTPM’s state is protected using a pTPM sealing key with no PCR constraints, which is securely held by a remote party. The remote party is responsible for keeping this sealing key secret; it must only be released to the vPlatform Manager upon request from an approved vTPM, and only after the platform has been appraised to determine whether it is trustworthy. (See launch discussion.)

In both cases, or in other cases not described here, the mechanism used to save the vTPM data securely must reflect what is described in the vEC. (10.2). In the case of the PCR-protected vTPM state, the particular pTPM PCRs chosen and their expected values (and any applicable update policies) must be included in the vEC. In the case of the remote party, the vEC must hold an identifier for the remote party, usually a public key. This allows an appraiser to evaluate whether the vTPM secrets are protected to a level they consider trustworthy.

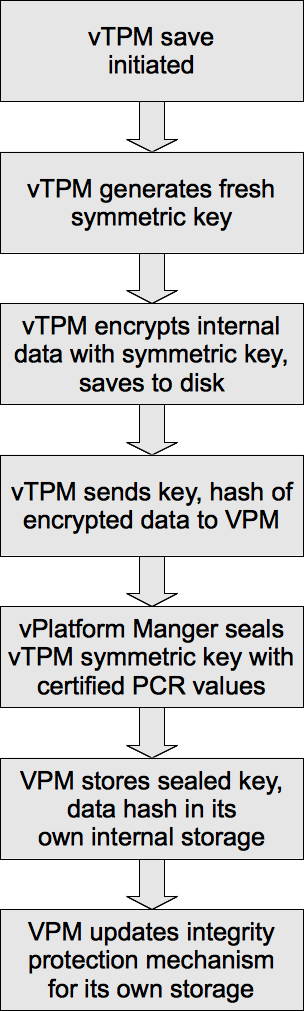
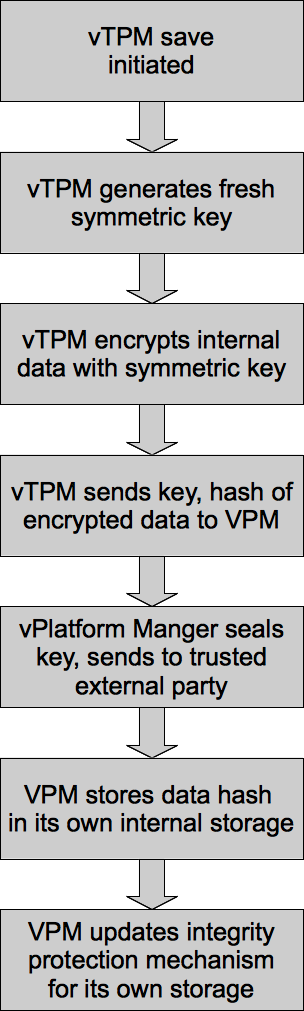
 

Figure : Two approaches to saving vTPM data. On the left, data is protected by locally encrypting with a pTPM key in such a way that it can only be accessed with particular pTPM state. On the right, data is protected by sending the data encryption key to an external trusted party, so that the data is not available without the external party’s approval.

Note that not every change to vTPM data should require a save. Any data which is volatile—which would be reset to a default state or erased upon reboot—may change freely without the state being saved. It is, however, essential to save data in the event of any changes which should not be rolled back, such as the vTPM being cleared, the vTPM ownerAuth changing, the contents of the vTPM’s NVRAM changing, and so forth.

One particular concern in saving state is timing-based attacks or crashes. Race conditions are easy to create in these architectures. If the machine crashes after the encrypted data has been saved but the hash has not been updated, the vPlatform Manager may falsely register a rollback attack. If the encrypted data is saved but the VDK has not been, the new data will be inaccessible. Worse, if the vTPM changes its state in a user-visible way before all of the save state actions are complete, a rollback attack is actually possible.

It is recommended that old data not be deleted until each component receives comfirmation from lower-level components that save-state actions have completed successfully. Although it does slow user response time, it is also recommended that vTPMs not return success or failure codes to the user for operations which trigger a state saving process until the save has completed successfully, thereby preventing a rollback on such critical non-volatile state as the owner authorization or a TPM clear. Because very few vTPM operations actually affect non-volatile state, the slow response time on these commands will rarely be problematic. The primary exception is applications which are heavy users of the monotonic counter; a system implementing a nested vTPM architecture (where a hypervisor running its own virtualized platform architecture runs inside a primary VM) should come up with an alternate race condition prevention mechanism.

## vPlatform Shutdown

### Shutdown Requirements (normative)

* The vTPM MAY go through a save procedure on shutdown.
* The vTPM SHALL perform normal TPM shutdown operations when a virtual platform is shut down.
* vTPM secrets SHALL NOT be exposed when the vTPM is shut down. Any unencrypted vTPM secrets used during vTPM operation SHALL be securely erased from all system resources using good information sanitization techniques, such as overwriting with zeroes.
* If the Virtual Platform is being suspended, the vTPM volatile state SHALL be saved. Time SHALL NOT advance for the vTPM while the platform is suspended; for example, the Tick Counter SHALL NOT advance.
* The system SHALL take appropriate action to ensure that vTPM PCRs can be verified against the vPlatform state when a virtual platform is suspended.

### Shutdown Discussion (informative)

There are two particularly security-critical scenarios that fall under system shutdown: suspension, and partial shutdowns.

In normal use, we expect an entire vPlatform to be shut down simultaneously, in which case the vTPM will simply shut down. If vTPM state is being saved whenever a non-volatile change is made, as we recommend, shutdown does not necessarily need to update the saved state.

In a suspension, the guest VM and other VMs on the platform are simply stopped; the software running in them should not, generally, even be aware that the VM is suspending or resuming. Time, from its perspective, has simply stopped. However, in the case of a vTPM, we do not want suspend to be as invisible. In particular, we need to guard against substitution or modification attacks, where the state of the suspended vPlatform components change without a corresponding change in the vTPM's relevant PCRs. This sort of attack could cause an appraiser to establish trust in a vPlatform state which they would not normally consider trustworthy.

One mechanism for preventing such attacks would be to have a vPlatform suspension cause the VMM to send the vTPM the hashes of each component as it pauses and is stored; the vTPM can store that state internally (for integrity checking in its own internal memory, not in PCRs or other externally-visible TPM storage) and save itself as a suspended vTPM. On resume, the VMM would boot the vTPM first, and then the vTPM would be provided with the hashes of each resuming component; any mismatch between the suspended state and the renewed state would cause the vTPM to enter a failure mode, potentially becoming unusable until the next fresh reboot or even permanently (akin to the pTPM's self-checks on a resume from saved state).

Partial shutdowns and crashes offer another significant challenge, again in the particular context of making sure that vTPM PCRs accurately reflect the associated vPlatform state. There is no single correct answer for how individual VMs that crashed and restarted or shutdown and rebooted should be reflected in the vTPM PCRs, as it depends on the requirements of an individual implementation. The answer may even vary based on which VM crashed. The important feature to maintain is that any appraiser looking at the PCRs should be able to trust the PCR contents. Possibilities include:

* comparing the original boot-time measurement to the new one and extending the PCR if it has changed, showing the entire past state of the component since the vPlatform has booted but allowing changes; this provides the most information but requires an appraiser to evaiuate a more complex measurement and not simply use a golden hash.
* extending relevant PCRs with a generic “This VM crashed and was reset” value; for compoenents where only the current state matters but where crashes should be detectable; or where we want to both minimize the burden on appraisers during evaluation of measurements but also want to be able to react to potential problems.
* resetting the PCR and extending it with the new boot value as though it were a fresh launch ; this hides information from the appraiser, but may be suitable for VMs with no long-term state, and is the simplest for an appraiser to evaluate..

Crashes of the primary VM will normally result in a reboot of the entire vPlatform.

## Backing Up and Restoring vPlatforms

### Backup and Restoration Requirements (normative)

* The virtualized platform MAY support the backup and restoration of vTPMs.
* If backups are supported, restoration MAY be supported only on the original platform, or also on other machines.
* If backups are supported, there SHALL be mechanisms in place to prevent the backup from being restored while the original vTPM still exists.
* If backups are supported, all secrets (such as vTPM keys) SHALL be encrypted and all data SHALL have integrity protection. The secrets SHALL only be released during the approved restoration procedure.
* If backups are supported, there SHALL be a way to determine which backup is most recent. There SHOULD be a mechanism to ensure that the backup is updated whenever critical vTPM state changes. If this mechanism does not exist, there SHOULD be a way to determine how recent the backup was.
* If backups are permitted, this SHALL be reflected in the vEC (Section 10.2). The vEC SHOULD include information about the restoration procedures.
* If the vTPM is not guaranteed to update its stored data whenever its state changes, then this fact SHALL be included in the vEC. The vEC SHOULD provide some mechanism of determining whether the vTPM has been rolled back to a previous state, such as having a trusted server report the elapsed time between the creation of a backup and its restoration.

### Backup and Restoration Discussion (informative)

Backup and restoration is an area where our security goals for vTPMs come into conflict. On the one hand, vTPMs should never have state rolled back, never exist in more than one place, and never have their secrets readable outside of themselves. On the other hand, in real-world contexts and particularly enterprise contexts, we care deeply about data recoverability and system resiliency. As such, backups are often a necessity.

Although it is possible for a vTPM to be partially backed up with a Maintenance Archive just as a pTPM can be, it is likely that many enterprises would rather store the entire VM image or serialized VM data. In this case, the system administrators are responsible for establishing a secure backup policy that balances the need for backups with the security requirements for vTPMs and other similar domains: that they be kept up to date, be unique, have their secrets protected when not in use, etc.. We recommend the use of a vTPM backup require the approval of a system administrator or other trusted party, and whenever possible, that the age of the most recent backup be tracked so that any rollback effects are at least bounded and can be reported to appraisers.

### Backups using trusted server key (informative)

In addition to being sealed to the pTPM PCRs or a trusted authority as described in section 9.6, a VDK may also be encrypted using a public key from an authority that is trusted to only decrypt the VDK if it is necessary to restore the vTPM from backup. This encrypted key and the associated encrypted vTPM data should be stored off of the system on a backup server; for maximal security, the server on which the data and encrypted keys are stored should not be the same system that holds the authority’s private key, and the backup authority key should be protected. (One way to securely implement such a protected key is to create a binding key in the trusted backup server’s pTPM.) The certificate for this key must be included in any certificates issued for this vTPM, so that appraisers may determine if they are willing to trust the system.

Additionally, in order to make certain that only the certified backup authority key is used, the public portion of the authority key must be verified before use. One option is that the backup authority public key should be hashed and stored in pTPM NVRAM, in a section that is writable only by the platform owner. The Virtual Platform Manager is required, in this scheme, to check the hash of the public key it uses for encrypting backups against this value in the pNVRAM before it is used. Alternately, the backup authority public key could be measured into a pTPM PCR during system boot (for example, by including the key in the Manager’s code); as long as the key is included in the TCB, any boot of the system which must pass pTPM PCR verification will only use approved authority keys. This second approach also allows an appraiser to confirm the identity of the backup authority.

It is critically important to note that without appropriate verification of the backup authority’s public key, the vTPM data could be encrypted and sent directly to an attacker! This is therefore one of the most sensitive operations in the lifecycle of the entire platform.

### Rollback Prevention Discussion (informative)

The approach to rollback prevention recommended by this specification, described in detail in Section 11, uses the monotonic counter to track the most recent state of the system, and has the vPlatform Manager maintain signed associations between current vTPM data hashes (and any other vPlatform Manager data which may need rollback protection) and the current value of a monotonic counter.

While the monotonic counter approach requires updating all stored vTPM HMACs every time any vTPM is saved, it is not expected to have a high performance impact, because even though each save may be burdensome in some implementations, vTPM non-volatile state changes rarely and the infrequency of save operations should minimize impact. Barring applications making heavy use of the vTPM’s monotonic counters or NVRAM, we would expect most vTPMs to not even need to be saved once on most boots. Any rollback prevention approach rooted in hardware will have similar burdens for use cases which update frequently, since pTPM hardware is much slower than software. The monotonic counter approach also allows us to avoid one of the major hazards of the more obvious rollback prevention approach, using the pTPM’s NVRAM.

Although the pTPM’s NVRAM could theoretically be used to store a simple hash of the vTPM’s data (or, more likely, a hash of the vPlatform Manager’s data, which contains the hashes of data for all vTPMs on the system), the pTPM’s NVRAM has a limited number of writes in its lifetime, and a system with frequent vTPM updates might burn out the NVRAM, rendering the TPM useless and preventing any further updates from having any rollback prevention at all. In contrast, the monotonic counter is required to be able to sustain a frequent (several times per second) update schedule for years.

## Software Upgrade

### Upgrade Requirements (normative)

* The Virtualized Platform SHOULD support the updating of its software without eliminating all trust previously established in the system
* It SHALL be possible for an appraiser to know both the boot-time state of the system TCB and the TCB state after any software updates applied since the last boot. It SHOULD be possible for an appraiser to learn the upgrade history, including past measurements, of any trusted components which would have been relied upon to protect system or vTPM secrets, due to the risk of past compromises resulting in secret exposure. This upgrade history MAY be stored locally, or MAY be available elsewhere, such as with an enterprise IT server.
* It SHALL be possible for an appraiser to learn the update policy of the system, and how that policy will affect any trusted components such as vTPMs. If migration is supported, the appraiser SHALL be able to learn the update policy of all systems the vTPM may migrate to.
* The upgrade policy for the system SHALL be included in the vPlatform Credential. No upgrade shall be performed that makes any existing vPlatform Credentials for vPlatforms on the system invalid unless the relevant credentials are revoked.
* The Virtualized Platform is NOT required to allow updates while the trusted components are running, although it is permitted to do so. If runtime software updating is allowed in such a way that resulting software state is not measured into vPCRs, this SHALL be reflected in the vPlatform Credential.

### Upgrade Considerations (informative)

Software upgrades are another area of tradeoffs between security requirements. Software is imperfect, and will always require patches and updates; on the flip side, when we report the state of a trusted component such as a vTPM to an appraiser, we need the appraiser to be able to meaningfully decide whether or not to trust that component, often over a period of time. Simply updating the software in the background may result in the appraiser's information being outdated; worse, a component could have existed in a trustworthy state for some period of time, then an untrustworthy state, and then upgraded again into a trustworthy state. While this might be acceptable in components with no secrets or security-critical state, this would result in a complete lack of trust in a vTPM; in its untrustworthy state, all of its secrets could have been compromised, and we could no longer rely on it to protect our data or authenticate our VMs. As a result, any system which supports software updates---which is to say, most of them-- will need to provide a mechanism for the appraiser to decide not only whether the component is trustworthy now, but also whether it was trustworthy in the past and whether it will remain trustworthy for the duration of their interactions.

There are two primary approaches to solving this problem.

The first is to provide a policy—for example, “the system will be updated within one week of when Enterprise IT Authority determines that a non-critical patch is required, or within one day of when the IT Authority determines a critical patch is required”—which the appraiser can evaluate, independent of the individual software updates. The advantage of this approach is that it scales well over time; however, most implementations of it will rely in the appraiser trusting some third party authority.

The second approach is to have the appraiser itself verify the update history of the machine. This is made more complicated by the fact that most appraisers probably need more than a simple list of, e.g., the hashes of all VMM versions used on this machine since it was created. Software that was perfectly acceptable to run one year might be completely unacceptable five years later, but if we don't know when upgrades were applied, we don't know whether it was in a vulnerable or reasonably trustworthy state for most of its history. Additional complexity does not even necessarily contain useful information, particularly if we are using measurements that vary as rapidly as hashes.

System administrators may wish to consider instead including details of their upgrade policies and history; for example, a team running a Microsoft VMM might have an upgrade policy which guarantees that their servers are updated every patch tuesday, and within three days of any urgent out-of-cycle patches. If the administrators are trusted, an appraiser might reasonably accept such a guarantee, especially if the current measurement reflects this week's expected version.

In some cases, multiple update policies or complex role-based update policies may apply. For example, a vTPM owner might be authorized to update the vTPM software (equivalent to updating the pTPM firmware), but the server operator's update policy would apply for changes to the VMM.

Runtime updates, where some software is updated while a vTPM or security-sensitive guest is running, are not covered in this specification. All updates addressed in this specification assume that the platform—either physical or virtual, as appropriate to the software being updated-- is rebooted after the update is applied.

## pPlatform Change

### pPlatform Change Requirements (normative)

* Implementations MAY support a change to some or all of the physical platform, either by restoring from a backup onto a new machine, or by supporting significant and potentially security-relevant changes to the existing hardware.
* Implementations MAY support the migration of virtual platforms from one pPlatform to another directly, without having to move it to a storage device first.

The following requirements apply if pPlatform change or migration is supported:

* Support for pPlatform change or migration SHALL be indicated in the vTPM’s Endorsement Credential (see 10.2).
* The Endorsement and Platform Credentials SHALL provide sufficient information for a verifier to determine whether they will trust both the current platform configuration, and possible configurations the vTPM and Virtual Platform may be migrated to. This MAY take the form of specific requirements or policies which all migration targets must meet. It MAY involve the approval of an external Migration Authority.
* Each vTPM SHALL only execute on a single platform at a time, and vTPM state SHALL persist across the platform transition.
* vTPM secrets SHALL be encrypted at all times during migration. vTPM secrets SHALL only be decrypted after migration if the destination platform meets the criteria described in the vTPM’s Endorsement Credential, and the associated Virtual Platform meets the criteria described in the Platform Credential.
* When VMs, including vTPMs, are migrated, the implementation MAY migrate only the data, or MAY migrate both data and code. In cases where a VM maintains no state, migration MAY consist of creating a new equivalent VM on the destination machine.
* The vTPM in a Virtual Platform SHALL always migrate with the Virtual Platform’s primary VM.
* Any VM which provides services to only a single Virtual Platform and which is measured into the Virtual Platform’s vTPM when it is launched SHALL migrate at the same time as the vTPM and primary VM.

### pPlatform Change Discussion (informative)

There are a number of use cases which result in a physical platform change and therefore a renewed binding between virtual and physical platforms. Most notable among them:

1. Replacement of the major physical components of the platform, such as a motherboard replacement. Generically called “FRU replacement”, it is usually the result of hardware failure.
2. Backup/restore of a vPlatform on a different pPlatform, or the direct migration of vPlatforms as part of a system upgrade. Generically called static migration, this is usually the result of one pPlatform being replaced by another due to machine loss or upgrade.

The pPlatform change section is meant to cover both of these. Live migration, the migration of Virtual Platforms for load-balancing or performance reasons, will be covered in a later version of the sepecification.

The problems that need to be solved are:

* Keeping secrets safe during the migration
* Proving to an external entity that vTPMs only exist on one machine after the migration.
* Letting an external entity establish long-term trust in a vTPM, knowing that it may migrate

This specification assumes:

* That the user is also the owner of the system
* That the system will prepare for static migration before it is needed.

### Migration and Backup Techniques (informative)

In this section, we describe three migration techniques that preserve the ability of the user to attest that his vTPM is not running on two different platforms simultaneously.

#### Local Owner Approval

This scenario is primarily aimed at upgrade situations, where a local machine owner wishes to upgrade hardware or otherwise switch to a new platform. These upgrades are presumed to happen rarely; for frequent upgrades, using the techniques described in later sections is recommended.

* The new system will provide to the old system a certificate for the new system that certifies the pSRK of the new system is on a genuine PC Client system with a genuine TPM.
* The old system will create a 256 bit AES key, and use it to encrypt all secret data for VMs that will be migrating. Non-secret data may be encrypted, or may be migrated in the clear.
* The old system will then create a Binding key with a parent of the pSRK of the new system, and use the public portion of that binding key to bind the 256 bit AES key.
* The old system will send a message to the backup or migration server (should there be one) that control of the system is being transferred to the new system with the new pSRK; in certain enterprise use cases, approval of the server may be required before the transfer proceeds.
* While it awaits confirmation from the server, the old system will send the encrypted copy of the data to the new machine.
* When confirmation of the server’s approval is received, the old system sends the binding key, which can be loaded in the new pSRK.
* When the new machine acknowledges that it has received the encrypted data and can load the newly created binding key, the old system sends the encrypted AES key and erases the local copies of the encrypted data and unencrypted AES key.
* When successful decryption has been confirmed, the old machine erases its copies of the encrypted AES key and the binding key for the new machine and sends a confirmation to the new machine. If permanently migrating the system, it clears its own TPM.
* The new machine issues an extended virtual Endorsement Credential for migrated vTPMs, including the original Endorsement Credential and certifying that the vTPM in question is now resident on the new machine and not the old machine.

#### Using a remote server to approve migrations

An external Migration Authority can be used to help in the migration and certify that the migration has occurred correctly. This agency can easily be the same agency that performs backup for the vTPM in the case of death of the pTPM on the system (usually when the motherboard dies, taking the pTPM with it).

The procedure may look very similar to that described above; the same structure of encrypted VM data and separately encrypted symmetric key we expect to be common to most migration scenarios. There are many variations of this approach as well; however, the common theme is that the Migration Authority plays a critical role in deciding when and where the migration should occur.

In the first variation, the Migration Authority closely parallels the Migration Authority described in the 1.2 specification for Certifiable Migratable Keys. Here, the symmetric key is encrypted to the Migration Authority; the Migration Authority can then re-encrypt the symmetric key for the destination system based on its internal policy. The encrypted data can be sent directly to the destination system, placed in a public location accessible to the destination system, or sent to the Migration Authority along with the key. It is worth noting, however, that the trust placed in the Migration Authority is significantly higher if the MA has access to both the symmetric key and the encrypted data; in that case, it is trusted not only to select only appropriate migration targets, but also to not decrypt and misuse VM data. The Migration Authority is also responsible for ensuring that the vTPM is only executed in one place at a time; for example, by appraising the old server to verify that is running the expected software with expected correct behavior before accepting its claim that the vTPM data has been deleted from the machine.

In the second variation, the Migration Authority instead generates the binding key that will be used to encrypt the symmetric key, and provides this key, signed, to the original server. Here, the protocol itself looks much like the owner-approved version, but the transaction must be approved by the Authority.

#### Using a local backup server

The previous scenarios share a common theme: migration from one working machine to another. However, it is quite common in the real world that we must be concerned with machine loss or failure. In these cases, we wish to set up a trustworthy backup system. These systems will normally be run by an enterprise IT department, although they can be set up using a hard drive and a USB drive with a RSA key on it if need be. We can separate their functionality into two parts: storage of encrypted backup data, and a Backup Authority (BA), which possesses a decryption key.

In the ideal case, backups would happen whenever a vTPM’s state is updated. When the state is encrypted with the VDK and saved to disk using the Save State procedure described in 9.6, the VDK is encrypted using the Backup Authority’s public key in addition to whatever mechanisms are normally used to save state. The encrypted data and BA-encrypted VDK are saved to the backup storage system, along with a timestamp to verify the freshness of the saved image. In the event of a system failure, the encrypted VDK is provided to the BA, which would then rencrypt it to an approved target machine. The freshly encrypted VDK and the most recent saved state would then be provided to the target machine, allowing the vTPM to launch again. Non-vTPM data can be backed up and restored in the same fashion.

However, remote backup on every system save may be overly burdensome for some environments. In this case, there is a clear tradeoff between frequency of backup and risk of rollback of vTPM data; it is up to the enterprise or other implementer to determine the best choices for their use case. Regardless of the choice made, the frequency of backup and rollback-resilience policies should be assessable by the appraiser. Normally, backup policy would be included (often by reference) in the Endorsement and/or Platform Credentials. If rollback is known to have occurred—for example, if a week’s worth of vTPM updates were lost—it is generally recommended that new credentials be issued which include this information or all users of the Virtual Platform be notified, so that users and communications partners can take appropriate action.

.

# Attestation and Remote Verification

## The Importance of Remote Verification (informative)

The purpose of this specification is to support the creation of trusted virtualized platforms. This raises important questions: trusted by who, and for what purpose? The answer varies. A platform trusted by a user to support his trusted banking application may not be trusted by an enterprise to download high-value data; a platform trusted to provide safe sandboxing by hotel guests checking e-mail wouldn’t usually be trusted to provide multi-level security for a government agency processing classified information.

This means that in order for a trusted virtualized platform to be useful, it must provide its users and communications partners with mechanisms by which they can verify that the platform—both physical and virtual—is in a state suitable for the use which the user or communications partner has in mind. The same user may have different requirements in different circumstances; a proper trusted architecture should not support mere blind trust, in which once we give trust in a machine it is always assumed to be good for all purposes. Instead, we need to provide information which allows anyone who relies on the platform to decide dynamically whether this platform is trustworthy for their purposes now, has been trustworthy for those purposes in the past, and is likely to be trustworthy for those purposes in the future. We therefore refer to remote verifiers—whether they are users or communications partners or anyone else seeking to determine the trustworthiness of the machine—as appraisers, since they are appraising the machine against their own set of approval criteria.

We refer to remote verification because local verification is an extremely difficult problem which requires specialized hardware support or major machine provisioning investment to even begin to solve: a compromised machine potentially has complete control over all calculations performed, so any verification method that does not use an external channel (such as user recognition of a secret picture) can be replaced with whatever results the compromised machine desires. True verification requires a safe place to stand outside of the compromised machine; hence, “remote verification”. The distance can be short; a USB dongle which performs a TPM-based attestation of the trusted virtualized platform and produces a green or red light upon verifying the results is still remote for our purposes, because it is separate from the platform itself.

Our primary techniques for remote verification are proper certification and attestation protocols. Certification lets us establish trust in critical keys and thus in the operation of critical pieces of software; attestation protocols let us determine the state of the machine at the time of the appraiser’s challenge, using the TPM and vTPM’s PCRs along with any other information that may be relevant.

## Virtual Endorsement Credential Requirements (normative)

* A vTPM SHALL have its vEK certified by the vTPM Factory (producing a vTPM Endorsement Credential, or vEC). It MAY have additional keys certified by its creator; the vEK MAY be certified by additional entities.
* The vEC SHALL be signed using the VFK, or by a CA acting as the Factory certifying authority. If the VFK is used, it SHALL be certified as a pTPM key, restricted to the vTPM Factory’s use, by a trusted authority. If a CA is used, it SHALL NOT issue a vEC without verifying the trustworthiness of the EK and its associated vTPM. Certifications SHOULD be in a standard X.509 format.
* The vEC SHALL contain:
  + information sufficient for an appraiser to decide whether to trust the vTPM, either directly from the credential or via separate attestation on the machine (see Attestation, Section 10.5)
  + information about the vTPM upgrade policy (including if there is no such capability)
  + information about the vTPM backup and restoration policy (including if there is no such capability, and if there is, which Migration Authority (see 8.6) is responsible for enforcing it)
  + information about the vTPM migration policy (including if there is no such capability, and if there is, which Migration Authority is responsible for enforcing it)
  + information about the vTPM version, and any additional capabilities such as deep attestation functionality
  + a reference to the platform credential
* The vEC SHALL be usable to certify other vTPM keys in the same way that a pTPM’s EC is.
* The vEC SHALL be stored in the vTPM’s NVRAM.
* The vTPM Factory or vPlatform Manager MAY issue credentials for AIKs

## vTPM Certification Discussion (informative)

vTPM certification is all about establishing trust in a particular set of keys, establishing a link between those keys and a vTPM, and providing an appraiser with the ability to determine that the vTPM is running approved software on an approved system. Certification criteria are deliberately vague, because different appraisers might have different needs. For example, one appraiser might want the precise sets of pTPM PCRs used to protect the vTPM, in order to verify the system software against its own internal set of approved versions. Another, likely part of an enterprise, would like a more abstract description of the system update policy and the public key of a software-signing authority, In all cases, the vTPM must be certified by the vTPM Factory, much as a pTPM should be certified by its manufacturer; the creator knows what under constraints the vTPM is operating, and can guarantee those constraints were present at creation. However, vTPM Factories are not human-monitored factories with standard procedures that can be vetted; they are software, often running on the same platform as the vTPM itself, and a verifier needs to have a reason to trust that software. As such, vTPM certificates are really multi-part: the certificate for the vTPM root key (vTPM Endorsement Credential, or vEC) and the certificate chain for the vTPM Factory certifying that it, too, is operating correctly.

In order to confirm that the Factory is trustworthy to create and certify vTPMs, we require two guarantees from its certificate chain:

* that the vTPM Factory was running and using approved software at the time that the vTPM was certified; and
* tthat the key which certified the vTPM was only available to the vTPM Factory. (This is necessary to prevent malicious components running on the same system as a trustworthy Factory from forging vTPM credentials; the constraint can be implemented using pTPM localities, constrained software access to the pTPM, access control mechanisms in the VMM, or any other mechanism; however, the pTPM PCRs should allow the access constraints to be verified by anyone who wishes to establish trust in the vTPM.)

We expect vTPM certificates to generally conform to a common template on a given platform. For example, the vTPM Factory might contain a template X.509 Platform Certificate that it can quickly fill out the few fields that are different for each VM (e.g. EK public key and subject alt name), then include a signature. The vEC is then signed using the VFK, which as described previously is a non-migratable pTPM Signing key whose public key has been certified using the TPM’s CertifyKey operation and a pTPM AIK, in order to provide verifiable proof of the platform identity and system state at the time the vEC was created.

This approach has the advantage of excellent speed and dependability, although its reliability and security will depend heavily on the isolation of the vTPM Factory and the enforcement of the constraints on access to the VFK. It also requires recipients to follow a longer than usual credential chain, since we need to verify certificates produced by the vTPM, certificates produced by the pTPM, and certificates certifying the pTPM. In enterprise environments where the complexity is a limiting factor, it may be advisable to have a CA whose role is to accept a complete vTPM AIK or Signing Key credential chain (including the vEC, vPC (described in the next section), and vTPM Factory chain), verify each element, and issue a new, simpler x.509 credential containing only that information that the enterprise considers relevant for internal appraisers.

Once the vEC is created, it is placed into the vTPM’s vNVRAM, where it can be used provide data to an attestee that allows him to evaluate the trustworthiness of the system and vTPM. Additional credentials such as the vPC (described further below), and vTPM Factory chain may be placed in the vTPM’s NVRAM, or may be provided to an appraiser via some other mechanism. Once the vEC is in place, the vEK can be used to establish trust in other vTPM keys as described in the main TPM specification.

Additionally, the vTPM vEC should include information about the vTPM’s upgrade policy, backup and restoration policy and migration policy, as well as a pointer to identify any relevant authorities or servers that may be relied upon for correct decisionmaking or operation.

## Virtual Platform Credentials

### Virtual Platform Credential Requirements (normative)

* The Virtual Platform SHALL provide Virtual Platform Credentials (vPCs) along with the vTPM credentials. The vPC SHALL be created by the vPlatform Manager when the Virtual Platform is first created, and signed by the VPCK.
* The locally created vPC MAY contain references to external resources, such as centralized platform specifications or enterprise-standard configurations, which define the platform as described in this section.
* There SHALL be a single vPC per Virtual Platform.
* The vPC SHOULD be stored in the NVRAM of the vTPM associated with the Virtual Platform.
* The vPC SHALL include:
  + a description of the possible VM types associated with this Virtual Platform, including which VMs are mandatory and which are optional
  + what communications channels are available between VMs within the Virtual Platform, and any security policy enforced on those interactions
  + what communications channels are expected between this Virtual Platform and other entities on the same physical platform, and any security policy enforced on those interactions
  + the expected contents of vPCRs used by this system
  + the localities assigned to VMs in the Virtual Platform, if any
  + any locality constraints the vTPM enforces on the vPCRs. If locality constraints are used, the vPC SHOULD provide the appraiser with information about how virtual localities may be remotely verified.
  + the nature of the vRTM, how it is measured, and by what component
  + any other information an appraiser may need to establish trust in vPlatform components based on vTPM contents, including supported deep attestation mechanisms
* Information in the vPC MAY be included by reference (e.g. a pointer to an external document location with a hash of its expected value) to reduce the length of the credential.

### Virtual Platform Credential Discussion (informative)

The primary role of a Platform Credential is to allow an appraiser to reliably interpret information provided by TPM PCRs. They are rarely used in pPlatform scenarios, because pTPMs (at least at the time of this writing) and the associated pPlatforms are all implemented according to the PC Client specification. This platform specification defines such things as which components will be measured into which pTPM PCRs, what locality constraints are imposed on which PCRs, which PCRs are resettable, and so forth. No comparable specification exists for virtualized platforms; and because of the wide variety of virtualization use cases, it is not clear that a single platform definition could meaningfully cover them all. This means that the Platform Credential becomes quite essential in virtualized systems.

The most essential role of the vPC is to allow an appraiser to confidently interpret the meaning of various vPCRs. Without the vPC, an appraiser who expects, for example, vPCR 26 to contain measurements provided by the vPlatform’s trusted measurement VM could be badly deceived by a system where vPCR 26 is actually a resettable PCR controlled by the guest. Mismatched platform expectations can also cause availability problems; for example, if an appraiser is evaluating a given vPCR’s contents assuming it to contain measurements of a networking helper VM when instead it contains a virtual DVD image, the system is likely to fail; even if the measurements might be good in some absolute sense, the system cannot reasonably be appraised if the contents cannot be interpreted properly.

## Attestation of Virtual Platform State

### Virtual Platform Attestation Requirements (normative)

* The system SHALL support attestation of the state of the guest VMs and virtual platforms by use of vTPM PCRs (i.e., vTPM quote).
* Guest VM attestation SHALL include evidence about the Root of Trust for Measurement for the guest, either in a vPlatform Credential, in the vTPM credential, or via some other remotely verifiable mechanism.
* The system SHALL support attestation in the form of the vTPM credential, providing a remote appraiser with information about both the trusted computing base (including VMM and vPlatform Manager) and vTPM at the time the vTPM booted. This information SHALL be provided to the appraiser along with all vTPM-rooted attestations.
* The boot state of the guest VM associated with a given vTPM SHALL be included in the vTPM PCRs. The boot state of any other VMs in the vTPM’s Virtual Platform SHOULD be included in the vTPM PCRs. Runtime measurements of VMs in the Virtual Platform MAY be included in the vTPM PCRs.
* Any VM which uses vTPM PCRs for state reporting SHALL be itself measured into a vTPM PCR. All measurements in vTPM PCRs SHALL be rooted in the Virtual Platform’s RTM, or in a component whose state is attested to in the vTPM credential.
* The attestation of any given Virtual Platform SHALL NOT be affected by the state of any other Virtual Platform, unless a deep attestation is being performed that covers service vPlatforms (see [10.6])
* The system MAY support privacy policies, such that attestations are only performed with trusted appraisers, so that sensitive information is not released, etc.

### Virtual Platform Attestation Discussion (informative)

Most Virtual Platform attestations, at a high level, will take the form of a vTPM quote with optional additional information, much the same way that most pPlatform attestations take the form of a pTPM quote with optional additional information. (Because this specification addresses trusted virtualized platforms, we are not concerning ourselves with purely software-based attestations, such as antivirus reports.) However, because of the complexities of a virtualized system, we can think about a vPlatform attestation as breaking down into three parts: attestation about the trustworthiness of the vTPM (usually in the form of its credentials, which provide evidence about the state of the system while the vTPM is running); attestation about the trustworthiness of the vRTM; and attestation about the software running in the Virtualized Platform. As with a pPlatform, our fundamental goal is to build a chain of trust from some reliable source (the pTPM) to the user-level software or other component which the appraiser cares about.

Normally, the vTPM attestation is passive; we present credentials which claim that the use of any vTPM key proves that the system is in an acceptable state, rather than actively presenting a pTPM quote of the current state. Some systems may want to allow the appraiser to request a pTPM quote in realtime as part of a vPlatform attestation, either because the vTPM credential does not provide sufficiently detailed information or because they believe the current measurements may have changed (for example, if the system supports runtime measurement of the VMM or vPlatform Manager); this is ***deep attestation***, and will be addressed in detail in Section 10.6.

The attestation of the vRTM connects the components which measure the vPlatform with the lower-level trusted components. Although this **can** be done using a distinct chain to the pTPM (and thus pRTM), it is normally built upon vTPM attestation; if the measurement of the vRTM is included in the vTPM’s attestation, and the appraiser believes it is good, then the appraiser can rely upon measurements produced by the vRTPM. Because the most common form of vTPM attestation is through the evidence in a credential combined with the use of the certified key, vRTM attestation will usually be achieved through the same certificates. A vTPM quote with appropriate credentials will both provide the vRTM’s measurement results and attest to the correctness of the vRTM itself. We discuss what vRTMs may be and options for measuring them in the next section.

The remainder of the attestation is built on the combination of vTPM quote and trust in the vRTM; a chain of trust is built from the vRTM to higher components, and measurements are stored in the vTPM PCRs, as supported by vPlatform software. Just as in a pPlatform, it is essential to connect all links in the chain of trust. vPlatforms, however, support much more varied chains of trust. For example, if my vPlatform contains a dedicated measurement VM performing VM introspection on my main guest VM, my attestation might include a standard boot-time chain of trust for the guest in low vPCRs, and a distinct set of high vPCRs (normally above the pTPM’s 24) establishing trust in the measurement VM, and then containing runtime measurements of the guest. The appraiser could evaluate both branches of the chain of trust before deciding whether or not it approves of the virtual platform.

### Virtual Platform RTMs (informative)

The RTM of the VM, since it is implemented in software, must itself be measured. One method for this is to have the vPlatform Manager, which itself is measured into the pTPM PCRs, place the measurements in the “static” vPCRs. In this case, the measurements will follow the PC client specification [14], just produced by the Manager,

A second way of doing this measurement allows the vBIOS to act as the vCRTM, just as the regular BIOS does would in a physical platform, wth one exception: the vCRTM would not measure itself, but rather be measured by the vPlatform Manager, which would place the vCRTM measurement in vPCR 0 before the vBIOS starts. This provides the same security as immutability would in the CRTM of a physical machine BIOS.

A third way of doing this measurement is to have the VMM or another service (which must itself be measured into the pTPM or vTPM) take a hash of the VM state before it boots, and store this in a vTPM PCR. Since vTPMs may contain more than 24 PCRs, a higher PCR not mistakable for any pTPM PCR is the recommended choice. If you take this approach, it is important to extend vPCRs 0-5 with a cap value, so that malware cannot extend them with deceptive ‘boot’ values. Note that if a Virtual Platform consists of multiple VMs, the same component may measure them all at boot; here, the VMM or trusted service is acting as the vRTM, but the chain of trust effectively branches, and it is even possible that each VM in the Virtual Platform may have its own chain of trust recorded in separate vTPM PCRs.

These methods can also be combined; for example, by having the main guest VM perform a “normal” boot from vBIOS up filling in the low vPCRs, augmented with a boot-time image measurement in a high vPCR, while helper VMs simply have their boot-time image measurements taken by the VMM and stored in separate high vPCRs.

### Simple Virtual Platform Software Attestation (informative)

The simplest form of vPlatform software attestation is to have the remote appraiser request a vTPM Quote. The PCRs in the vTPM will reflect the launch state of the Virtual Platform; the platform credential can be used to determine what the expected chain of trust is from the vCRTM through the primary VM measurements and any additional VM measurements expected in the platform. The appraiser, in addition to checking this chain of trust, would also verify the vTPM’s credentials, using the vEC to establish trust in the vTPM itself and therefore in the contents of the quote.

### Complex Virtual Platform Software Attestation (informative)

A more complex vPlatform attestation might use both the vTPM and specialized measurement VMs to perform a runtime assessment of the system. The specialized measurement VM might be as simple as an antivirus, or might be as complicated as a deep kernel memory inspector. Measurement VMs are particularly powerful in trusted virtualized platforms when modern VM introspection technologies are used.

For example, a remote appraiser would issue a request for a guest VM measurement along with a fresh nonce and a PCR mask.  The measurement agent would perform the VM measurement and store the measurement along with the nonce and the measurement request in an appropriate resettable PCR that only this measurement agent should have access to.  The agent would then request a vTPM quote using the given nonce and PCR mask, which should include the resettable PCR the agent uses. This would be passed to the appraiser along with the full measurement report.

If there are vTPM PCRs containing a measurement of the measurement agent, and the appraiser uses an appropriate mask, the appraiser may check (1) the validity of the vTPM quote, and (2) that the measurement of the measurement agent meets its standards. If the appraiser trusts the integrity of the vTPM, then the passing of these checks gives justification to the appraiser in believing that the VM measurement it receives is a genuine measurement, and allows the appraiser to establish trust in the measurement VM. It may then evaluate the contents of the measurement report, with assurance that the VM that produced it was also trustworthy.

More complex attestation protocols with additional features also exist; however, they are out of scope for this version of the specification.

## Deep Attestation

### Deep Attestation Requirements (normative)

* The system SHALL provide a mechanism to allow an appraiser to evaluate the current state of the pTPM. This MAY be through a realtime attestation of the pTPM, normally via intermediary components, or MAY be through credentials (such as TPM\_CertifyKey certificates) providing a verifiable set of claims about the state of the pTPM when certain operations are performed.
* The system SHALL provide a mechanism by which the pTPM state can be reliably associated with a given vTPM attestation. This mechanism SHALL guarantee that the vTPM is running on the pTPM’s platform, at the same time (within acceptable margins) as the pTPM attestation is performed or certified state claims apply, during the same boot as the pTPM attestation is performed or certified state claims apply.
* The system MAY support deep attestation which associates the state of multiple vPlatformss; for example, if a service vPlatform exists which provides networking services to all other vPlatforms on the system, a deep attestation service could be provided which provides evidence about the current state of the service vPlatform and associates it with an attestation from a user vPlatform.
* Any deep attestation protocols and services SHOULD support privacy controls, and SHOULD NOT allow the association of arbitrary vPlatforms on the system without the approval of the system owner. This is not in conflict with the previous goal; the amount of information provided to an appraiser SHOULD be calibrated to the needs of that appraiser to trust other vPlatforms on the system.
* Any deep attestation protocols and services SHALL be designed to mitigate against the following threats:
  + Man-in-the-middle attacks (associating a vTPM with the wrong pTPM)
  + Replay attacks (associating a vTPM with measurements of the pTPM at a different time)
  + Modification attacks (changing the values of otherwise legitimate messages in such a way as to deceive an appraiser)

### Deep Attestation Example Implementations

The goal of deep attestation is to tie the quote of a vTPM to the current state of the pTPM, and in some cases, to the current state of another vTPM. Deep attestation requires us to make tradeoffs between the security of the protocol, the simplicity of the protocol, and the complexity of our platform. In particular, there are two major challenges of deep attestation:

* proving the relationship between the multiple components being attested to, and
* proving that those attestations happened at the same time.

In the following scenarios, we will present several deep attestation examples in detail, and discuss what the advantages and disadvantages are for each.

It is important to note that because deep attestation is fundamentally about tying together multiple components on a platform, deep attestation techniques are *extremely* implementation-dependent. A protocol which provides all of the essential information in a secure fashion for an extremely simple architecture of single-VM virtual platforms booting in a known order might leave out security-critical information for a complex architecture of multi-VM virtual platforms booting in a random order and providing services to each other; and a deep attestation protocol designed for the second architecture is likely to require components that don’t even exist in the first. When selecting a deep attestation approach, the key questions to ask are:

* What components’ state do you wish to associate? The answer is usually “a vTPM and the pTPM”, but not always; “A vTPM, its pTPM, and the state of the trusted-path-to-user mechanism running in another vPlatform” is an example of a more complex association.
* What threats are you concerned about? A deep attestation protocol meant merely to provide further detail about a trusted machine can make assumptions that a protocol intended to test the current trustworthiness of critical software cannot.
* Does your protocol merely require the reporting of existing measurements, such as TPM quotes? Or do you require fresh measurements of some components? (Note that such measurements are not possible in many architectures, but in systems running for an extended period of time, boot-time measurements may not be sufficient to establish trust.)

#### Network-Visible Attestation Manager

In this scenario, only two attestations (or quotes) are needed to attest to both the VM and the VMM/services. Here, we assume that the vTPM comes with information in a credential—which might be the main vEC, or might be a specialized credential—associating the vTPM’s root key or identity key with a particular underlying pPlatform, and providing a pointer to a separate interface—normally a network-visible Attestation Manager VM—where the appraiser can go to obtain attestation data about the underlying VMM of the VM. This credential must, of course, be signed with a key that itself is certified by a known authority, or is a known authority; and must include information which cryptographically identifies the pTPM associated with the pPlatform (normally an AIK public key).

The appraiser in this scenario would perform a normal vTPM quote request protocol, then refer to the credential pointer, and follow up with an attestation request for the pTPM, via the Attestation Manager. Note that we **must** provide the appraiser with a mechanism by which they can cryptographically verify the association between the pTPM and vTPM; this might be via the vTPM’s normal certificates (if the AIK used to perform the quote is also used to certify the VFK, for example), or might be via an Attestation Manager mechanism which extends vTPM and Virtual Platform identifying information (such as public keys or measurements) into a resettable pTPM PCR. In the second case, of course, the request to the Attestation Manager must include identifying information about the virtual platform; this identifying information should be cryptographically bound to the vTPM (e.g. a vTPM key), to prevent untrustworthy vPlatform A from claiming that it is actually trustworthy vPlatform B.

#### Deep Quote Command Description (informative)

Deep Quote is a deep attestation command given to the vTPM which produces a combined output of a vTPM quote and a pTPM quote. Structurally, Deep Quotes are very similar to the pair of TPM quotes that would result from the Network-Visible Attestation Manager approach; however, they do not require that any components in the Trusted Computing Base be network-visible. On the flip side, the Deep Quote command, like the Ephemeral AIKs described below, Deep Quote requires new functionality be supported by the vTPM.

The core idea of the Deep Quote command is that a requestor, instead of simply requesting a vTPM quote, requests a paired quote. The requestor would provide a single nonce and two PCR masks: one for the vTPM, the other for the pTPM. The vTPM would generate a quote as usual; but instead of passing the quote back to the user, the vTPM would provide it along with the nonce and the pTPM PCR mask to the vTPM Manager, as part of a request for a Deep Quote. The vTPM Manager—which already has a secure and authenticated channel to the vTPM, which it uses to provide the vTPM with its VDK and receive vTPM state updates—will identify the vTPM, and use one of the pTPM’s resettable PCRs to store the measurement of the vTPM that was taken when the vTPM was most recently launched. It will also extend the resettable PCR with the vTPM quote, allowing the recipient to tie the vTPM quote reliably to that pTPM. The Manager will then perform a pTPM quote, using the provided nonce and PCR mask; if the PCRs are properly selected, this quote will provide evidence about the state of the VMM, vTPM Manager, vTPM, and any other relevant components of the platform’s Trusted Computing Base. The pTPM quote along with the pTPM PCR contents is passed back to the vTPM, which forwards them along with the vTPM quote back to the requestor for verification.

Because the vTPM Manager measurement is included in the pTPM PCRs, an appraiser with a deep quote can decide whether it trusts the Manager to correctly store and extend the vTPM measurement. If so, it can combine the TCB measurement and vTPM measurement to decide whether it trusts the vTPM quote, and thus the vTPM quote’s contents.

The Deep Quote command can be combined with complex attestation protocols that would normally use a standard TPM quote. For example, in the protocol described in section 10.5.5, the Quote command can be replaced with a Deep Quote; two masks, instead of one, would be provided; two quotes, instead of one, would be returned; and both physical and virtual TPM PCR contents would be forwarded to the appraiser for verification. The protocol is otherwise identical.

#### Deep Quote Command Requirements (normative)

IF the Deep Quote command is supported:

* The vTPM SHALL provide a clear command API which accepts the at least the following inputs:
  + One nonce field (also known as “user data”), as required by a TPM quote API (the TPM\_Quote or TPM\_Quote2 commands, or 2.0 equivalent)
  + Two PCR masks in the format required by a TPM quote API, one for the vTPM and one for the pTPM
  + An appropriate vTPM key, as required by a TPM quote API
* The vTPM command API MAY accept a pTPM key, as required by a TPM quote API; or the system may support using a standard owner-chosen key, to prevent the requestor from needing to have access to pTPM keys or key locations.
* The vTPM command API SHALL return at least the following outputs:
  + One vTPM quote, using the user-provided nonce in its nonce field, including the PCRs described by the vTPM PCR mask, and signed by the vTPM key
  + One pTPM quote, which includes the vTPM quote and vTPM measurements extended into a resettable PCR, and all other PCRs described by the pTPM PCR mask, and signed by the pTPM key. This quote MAY use the user-provided nonce in its nonce field.
* The vTPM command API SHOULD additionally return the following outputs:
  + The measurement of the vTPM, for verifying the extended resettable PCR
  + The current values of the pTPM PCRs, to allow the appraiser to verify individual PCR values rather than a single hash of the entire current state
  + If a pTPM key is not provided by the requestor, any applicable credentials to allow the appraiser to establish trust in the pTPM key used to sign the pTPM quote

#### Ephemeral AIK Description (informative)

[Diagram]

Ephemeral AIKs (eAIKs) are one method used to show the binding between a vTPM and a pTPM. In systems that use them, eAIKs are generated whenever the VMM / services layer is potentially changed in a way that would change its PCRs, e.g. by reboot. eAIKs are eliminated whenever the VMM terminates or is reinstantiated or a VM is (re)instantiated. An eAIK is generated for each vTPM. (Note: Their creation may be delayed until the vTPM is also called for). They are generated by the vPlatform Manager and placed in the private virtual Non Volatile Random Access Memory (NVRAM) of the vTPM (in the same non-user-editable portion as the EK and SRK), with the indicator set that makes them an owner evict key. PCR 16 of the pTPM is reset, and then the public portion of the eAIK is extended into PCR 16. At this point, the vPlatform Manager asks the pTPM to quote all the pTPM PCRs with its pAIK, creating a linkage between the pAIK, its PCR values, and the eAIK. We call this the eAIK Certificate.

The eAIK Certificate, along with any supporting information necessary to prove to an attestee that the VMM and its services are in a good state, is placed in the vTPM’s NVRAM. This supporting information includes the pAIK’s certificate and may include, for example, any other measurements normally included in a TNC proof of health.

When the vTPM attests to its health to a third party to retrieve its private data protection key (see 9.1.2.3), it will use its eAIK to sign a quote of the vTPM’s PCR state, provide any other information the third party requires for a proof of health, and also include the data read out of the vTPM NVRAM.

Some points to keep in mind:

* The actions of the CreateEAIK ordinal are similar to those used to create an EK. The key is immediately stored in NV space (TPM\_PERMANENT\_DATA) and has a fixed, well known handle. It is not wrapped by the SRK. No corresponding delete ordinal is required. It is sufficient to overwrite the old value.
* Ephemeral AIKs are NOT appropriate for use in scenarios when the primary threat that deep attestation is meant to address is software compromised after the initial boot-time measurements. Untrustworthy vTPM Managers, resulting from runtime compromise, could decline to reissue eAIKs or revoke old ones, leaving earlier measurements as the only ones visible to the appraiser; therefore, eAIKs cannot be considered a reliable source of post-boot measurement changes if compromise of vTPM Managers or vTPMs is a threat. However, they are a highly efficient solution for scenarios where the primary concern is that the appraiser be able to determine which particular software a machine is running, within an approved trusted set included in the Virtual Platform credentials.

#### Ephemeral AIK Requirements (normative)

If Ephemeral AIKs are implemented, the following requirements are in place:

* An eAIK SHALL NOT be saved when vTPM data is saved.
* A new eAIK SHALL be created when the vTPM is instantiated/reinstantiated, before the vTPM is populated.
* eAIKs SHALL be deleted whenever relevant pTPM PCRs change. The indexes of the pTPM PCRs whose values are monitored SHALL be included in the eAIK’s credential.
* When an eAIK is deleted because of pPCR change, a new eAIK SHALL be created, with a credential reflecting the new pTPM PCR state.
* The pTPM PCRs whose value changing will force a change in the eAIK SHALL include the following:
  + Any pTPM PCRs containing measurements of the VMM, or of components the VMM relies upon
  + Any pTPM PCRs containing measurements of the vTPM Manager, or of components the vTPM Manager relies upon
  + Any pTPM PCRs containing measurements of other components the vTPM relies upon
  + Any pTPM PCRs containing measurements of components that other VMs in the vTPM’s Virtual Platform may rely upon, which are not measured into the vTPM itself.
* The pTPM PCRs whose values are monitoried eAIK MAY include any and all other pTPM PCRs.
* eAIKs SHALL be deleted if the vTPM or TCB software is modified, even if this would not cause a pTPM PCR value to change.

If Ephemeral AIKs are implemented, the vTPM SHALL support the following ordinals:

**TPM\_CreateEAIK**

1. Validate the keyInfo parameters for the key description.
   1. The key length SHALL be a minimum of 2048. For interoperability, the key length SHOULD be 2048.
2. The other parameters of keyInfo (encScheme, sigScheme, etc.) are ignored.Create a key pair called the eAIK, “ephemeral AIK key pair”, using a TPM-protected capability.
   1. The type and size of key are that indicated by keyInfo.
   2. Set encScheme to TPM\_ES\_NONE and sigScheme to TPM\_SS\_RSASSAPKCS1v15\_SHA1.
3. Store the eAIK at key handle vTpmEaikHandle, specified in the Virtual Platform Specification.
4. Return the eAIK TPM\_PUBKEY.

**TPM\_RevokeEAIK**

1. Delete the eAIK at key handle vTpmEaikHandle, specificed in the Virtual Platform Specification.

If Ephemeral AIKs are supported, the following system provisioning steps SHALL be performed before any vTPMs are created:

* Create a pTPM identity key pAIK
  + No PCR constraints are required
* Get a pAIK certificate pAIKCertificate from a CA
* Create a pTPM signing key pSK
  + Lock pSK to trusted (current or recommended) PCR values, including measurements of the VMM and vTPM Manager
  + (This action should be repeated whenever a new set of trusted PCR values are established)
* Certify the key pSK using the pAIK and TPM\_CertifyKey

The pSKCertificate will include its PCR values

<informative comment> Since pSK use is locked to VMM PCR values, it is implicitly revoked when the VMM PCRs change, either during use or on reboot. A vTPM eAIK cannot be certified when the VM is in an unapproved state. </informative comment>

Whenever the vTPM is launched, the vTPM Manager is additionally responsible for performing the following actions:

* Create the ephemeral identity key eAIK using TPM\_CreateEAIK
* Create an X509 vCertificate for eAIK using pSK
* Define a vTPM NV index and add the vCertificate, pSKCertificate, and pAIKCertificate.

**VM reboot / resume:**

The eAIK and certificate chain are in vTPM NV space. No provisioning is required.

**VMM reboot:**

It is important that the certificate chain accurately reflect the VMM state. Therefore, the current VMM state must be reflected in the PCR values via physical machine reboot, DRTM use, etc. Revoking the pSK is unnecessary for a VMM reboot that does not affect PCR values, as a new pSK with the same PCR values could be created; however, pSK revocation is appropriate for cases where the pTPM PCR values should change to a new trusted state.In the event of pSK revocation, the vTPM manager must rerun the eAIK provisioning, which generates a new eAIK and certificate chain, deleting the previous ones.

**Reboot of pPlatform** – A reboot of the physical platform will cause renewed measurement of the platform. Because the physical and virtual platforms are bound together, a reboot of the physical platform will drive steps for a renewed binding.

If eAIKs are used, the architecture of the vTPM should guarantee the elimination of the eAIKs, in the event of a platform shutting down. Upon reboot, the vPlatform Manager is responsible for creating a new eAIK and populating the vTPM vNVRAM with the appropriate information about that eAIK as described in section [4,2.3.2.2](#_Ephemeral_AIKs), resetting the vPCRs and repopulating as appropriate.

The vPlatform Manager is also responsible for checking that a rollback of vTPM secrets has not occurred as described in section [4.2.7.1.1](#_Rollback).

**Reboot of a vPlatform** – In the event that the vPlatform is rebooted, and the pPlatform is not rebooted or changed in such a way that it must be reflected in an eAIK, it is not necessary for any eAIKs belonging to this vTPM or their information to be recreated. The SRTM vPCRs, however, need to be reset and repopulated as appropriate

**Migration Credentials**

Because a new eAIK is generated on each boot of a system, old eAIK credentials become useless upon reboot of the system. Therefore there is no impact on any eAIK credentials. The need for security in the process is against any vTPM persistent secrets (such as the SRK) or things stored by them. It may be possible to mitigate this risk by having the end user store intermediary keys between the SRK and keys and secrets that he uses, storing those keys doubly encrypted, so that another key is used to load such a key in the vTPM each time it is used. This provides security through the key split during disuse of the key and security of the eAIK attested VMM during usage of the keys. Of course this begs the question to some degree – where was the key stored that is used to decrypt the intermediate key? But if the VM being used is remote, then the local computer may have the answer to that.

### Privacy and Deep Attestation (informative)

One of the big challenges of deep attestation is privacy; not just the protection of personal data, but the prevention of accidental leakage of information to unauthorized parties in enterprise contexts as well. Whenever multiple vPlatforms are in use for distinct purposes, there is the possibility that the system owner may not wish all of those purposes to be visible to all communications partners. For example, if one vPlatform is a low-security sandbox for running untrusted programs, providing information about which banking software is in use in a higher-security vPlatform is often undesirable, as it could open the user up to phishing or hacking attacks. Similarly, in an enterprise Bring Your Own Device context, identifying the enterprise vPlatform could let an attacker find employees of a particular enterprise to target.

This kind of cross-vPlatform information leakage is even more critical when dealing with platforms that have multiple users; not only cloud servers, which we don’t address in this specification, but low-level servers that may provide services to several people at once. In this case, we may wish to prevent not only external communications partners but VPlatforms on the server itself from retrieving information about other running VPlatforms.

On the flip side, however, we sometimes need to provide information about other VPlatforms and the underlying platform. If the system has helper VPlatforms providing services such as networking to all VPlatforms, then each vPlatform has a good reason to want to verify or prove the proper operating of the networking vPlatform. And the existence of side-channel information leakage attacks between VMs in cloud servers means that in high-security scenarios, we may want highly trusted VPlatforms or appraisers to be able to verify all software running on the machine regardless of which vPlatform it operates in.

Deep attestation protocols are always going to be balancing between providing useful information and protecting privacy. Even the simplest of deep attestation protocols, associating a pTPM’s state with a vTPM, provides information about which software the physical platform is running, and the physical platform’s identity. This is why in many use cases, deep attestation protocols will want to have privacy-protection capabilities built in. For example, a deep attestation protocol could require the appraiser to prove their identity before the protocol is executed; the local software would then evaluate that identity against a list of approved parties before proceeding. A more complex protocol could perform an active negotiation, in which an appraiser requests specific information about the system, and the system decides whether the appraiser should have access to some, all, or none of that information before responding.

In this specification, we are deliberately vague about how these concerns should be handled, since they will vary dramatically with use case. However, they are important to consider before implementing your deep attestation mechanism, since insufficient privacy protection can actually be a major security risk, while overly restrictive privacy protection can result in a system no one is willing to trust.

# Appendix A: Rollback Protection Using the Monotonic Counter (optional)

## Rollback Protection Mechanism Summary (informative)

It is critical for vTPM data and vPlatform Manager data to be resilient against rollback attacks. Rollback attacks could cause changed authorization values to be restored to old values, deleted keys to be recoverable, and monotonic counters to violate their always-increasing constraints. However, the most intuitive way to do rollback protection—save a hash of the relevant data in write-limited pTPM NVRAM- suffers from serious burnout problems. Today’s pTPMs have a very limited guarantee of how many write operations they will support in their NVRAM; once those operations have run out, the NVRAM becomes useless, opening up the system to attack.

The mechanism described in this appendix uses the pTPM’s monotonic counter as an alternate rollback protection mechanism. While this approach is more complex to implement than simple hash storage, it is extremely burnout-resistant.

In brief, this approach works by taking advantage of the fact that the monotonic counters in the pTPM have values which can only increase. Each time data is saved, the vPlatform Manager increments the counter, and signs a rollback protection value for each piece of stored data—including all old data which is up-to-date based on the original counter value-- which contains both the hash of the data and the new value of the monotonic counter. When retrieving data, the Manager confirms that the current monotonic counter value matches that of the signed blob, and that the signed blob was created by itself in a trustworthy state. Any mismatch in the monotonic counter value shows that either the blob is not current, or an adversary has managed to increment the counter. In either case, preventing boot until a human can be alerted is the correct response. We do not dictate a recovery policy, although it is recommended that most enterprises have one, since the correct response varies based on the enterprise’s priorities.

## Provisioning Requirements (normative)

If the monotonic counter rollback protection mechanism is implemented:

* A rollback monotonic counter (RMC) SHALL be created in the pTPM. The vPlatform Manager SHOULD have exclusive access to this counter; otherwise, an adversary could cause false alarms for rollback attacks. (See the end of 8.2.2 for related discussion.)
* A Rollback Signing Key (RSK) SHALL be created in the pTPM. The RSK shall be a non-migratable SHA1, DER, or SHA256 (TPM 2.0 only) signing key.
  + The RSK MUST only be usable by the vPlatform Manager; this constraint can be imposed using locality, authorization values, limited access to the TPM driver, or other mechanism.
  + The RSK SHOULD be PCR-constrained; the pTPM PCRs SHALL contain measurements of the VMM and vPlatform Manager, and any other components relied on for the trustworthy execution of the vPlatform Manager.
  + The RSK MAY be replaced over the course of the platform's lifetime; in all cases, the VPCK SHALL continue to be PCR-constrained as described above if it was originally, even if the values change.
  + The RSK SHALL NOT be trusted to sign TPM internal data, such as quotes or CertifyKey certificates.
  + The RSK MAY be the same key as the VPCK.

## vPlatform Manager Save State Requirements (normative)

* Whenever vTPM data is saved, the VDK and a hash of the data is sent to the vPlatform Manager. In addition to storing this hash and the encrypted VDK, the vPlatform Manager SHALL:
  + Determine the current value of the RMC (M1)
  + Increment the RMC, producing a new value (M2)
  + Verify that the signed rollback data was signed by the RSK
  + For each HMAC in the rollback data, verify that the HMAC was produced by combining the hash of a VTPM’s state and the original monotonic counter value M1. Additional values such as a hash of the vTPM’s VDK MAY also be included.
    - If any HMAC fails the check, a rollback, substation, or data modification attack may be in progress, and appropriate action should be taken.
  + For the newly saved vTPM data and each stored vTPM which passed the last check, create an HMAC of the hash of the vTPM data and the current counter value M2; it MAY contain additional values such as a hash of the VDK for that vTPM.
  + Sign the new collection of rollback data with the RSK
  + Save the signature as part of the vPlatform Manager data. The rollback data itself MAY be saved, or MAY be reconstructed when the signature is checked.

## vPlatform Manager Load State Requirements (normative)

* Whenever a vTPM seeks access to its data, the vPlatform Manager SHALL do the following in addition to ensuring that the requirements in Section 9.4 are met :
  + Check the current value of the RMC
  + Check the signature of the rollback data to make sure that the blob was signed by the RSK
  + If the signature check passes, verify that the HMAC for the desired was generated from both the hash of the vTPM data and the current RMC value; it MAY contain additional values such as a hask of the VDK for that vTPM
  + The decrypted VDK SHALL only be sent to the vTPM if the signature check and HMAC check pass, in addition to all other constraints for VDK access being met. If the decrypted VDK is sent, the hash of the vTPM data SHALL be sent, so the vTPM can verify the integrity of its stored data.

## Other Rollback Protection Requirements (normative)

* The vPlatform Manager SHOULD follow the same process of combining a data hash and the current RMC counter whenever integrity protection and rollback protection is required for data stored in the Manager. Such data SHALL be hashed, and then treated as vTPM state data as described above.

1. Although Virtual Platforms without vTPMs can certainly exist, they are not trusted Virtual Platforms, and are not addressed in this specification. [↑](#footnote-ref-1)