Using the Windows 8 Platform Crypto Provider and Associated TPM Functionality

Functionality, Usage Models, and Reference Implementation

White Paper

Stefan Thom, stefanth@microsoft.com
Jork Loeser, jloeser@microsoft.com
Ron Aigner, raigner@microsoft.com
Paul England, pengland@microsoft.com
Rob Spiger, rspiger@microsoft.com
Jim Morgan, jimorg@microsoft.com

Version 1.0

Table of Contents

Introduction to Attestation and PCP-Kit 1

Scope 2

PCP-Kit and TPM Versions 2

Key Concepts Used in This Paper 2

Acronyms and Abbreviations 6

TPM Provisioning and Management 7

Pre-Windows 8 Architecture for TPM 1.2 Provisioning 7

Auto-Provisioning 7

Provisioning Through the UI 8

TPM State in the OS 8

Provisioning with WMI 8

Provisioning Differences Between TPM Versions 1.2 and 2.0 12

Windows 8 Certified Hardware Requirements 12

Platform Crypto Provider in Windows 8 13

Certificate Enrollment with the Platform Crypto Provider 13

Tracing Provider TPM Commands 14

BCrypt RNG Platform Crypto Provider 14

NCrypt RSA Platform Key Storage Provider 15

Executing Custom TPM Commands Through the TBS API 23

TPM Resource Virtualization 23

Command Filtering for 1.2 and 2.0 24

TBS API 24

Creating TPM 1.2 and 2.0 Contexts 24

Deleting TPM 1.2 and 2.0 Contexts 26

Obtaining the Windows Boot Configuration Log (WBCL) 26

Invalidating the System Trust State 26

Obtaining the TPM Version 27

Submitting a Custom TPM Command 28

Windows Boot Configuration Log 30

Windows Integrity Measurements 30

Root of Trust Overview 33
Platform Trust Considerations across Hibernation and Resume 33
ELAM Driver Data Measurements 35
Automatic Key Certification for Platform-Bound Keys 36
Format of the Key Attestation Data 36
Attestation API Reference Implementation 38
Introduction 38
Creating Attestation Identity Keys (AIKs) and Forming Remote Trust 38
Obtaining and Parsing Platform Configuration and Measurements 41
Platform Attestation and Validation 42
Key Attestation and Validation 46
Key Hostage 49
Overview of the PCP-Kit Package 52
PCPTool 52
Commands 54
Scenario Scripts 62
Certificate Enrollment Templates 66
PFX Private Key and Certificate Import 70
Windows Attestation Scenarios 71
Enterprise Asset Management with EK Certificates 71
Retirement of User Names and Passwords for Web Authentication with Mutual SSL 72
Remote Platform Attestation for Malware Detection 73
Platform Health Certificates 74
Certificate Enrollment with Key Origination Proof 77
Secure Key Roaming 78

Introduction to Attestation and PCP-Kit

This paper describes how a software provider can use the Microsoft® Windows® operating system and the Trusted Platform Module (TPM) to provide more reliable reporting of the health or policy compliance of computer systems and strong attestation of key origin and key properties. It also describes core operating system (OS) features for creating and using TPM keys that are bound to the physical machine, and how provisioning and other actions are performed. Finally, this paper describes a package of sample code and utilities called the Platform Configuration Provider Helper-Kit (PCP-Kit).

A Trusted Platform Module forms the low-level protected Root of Trust for Windows. The TPM can be a discrete cryptographic processor that is physically attached to the motherboard or may be an integrated implementation that provides similar security properties. One of the key capabilities of the TPM is to allow the authoritative reporting of the software running on the platform. This capability is called TPM-based attestation.

Many enterprises check software state and OS policy compliance before allowing computers to access corporate network resources. The goal of these checks is to ensure that the OS is properly patched, the OS configuration meets company policy, and that antivirus software is up-to-date. Unfortunately, in today's systems, this reporting is not very reliable because a genuine statement of system health can be spoofed by a rootkit or other malware. Attestation can provide a much more reliable anchor of trust for all online activities. Attestation uses the TPM to provide a cryptographically strong description of the platform configuration: With attestation, malware has nowhere to hide.

One of the central challenges with making attestation practical is developing policies for operating system components and settings that are useful. In particular, if reporting is too detailed (for example, the system provides details about every OS component and every security-sensitive OS setting), then it is hard to interpret which states are safe and which are not.

The approach built into Windows is to measure core OS components (which seldom change) and a specially vetted driver that is responsible for checking that the system meets policy. This specially vetted driver—called the Early Launch Anti-Malware (ELAM) driver—commonly checks for malware. .

Additionally, antivirus software is typically structured to include a core-detection engine (which also seldom changes) and a virus definition file that changes relatively often. Platform trust depends on both these components, so such configuration files are also included in the platform measurements.

The system firmware and the Windows OS record integrity measurements during boot in the TPM and maintain a log of the measurements in memory. For attestation, third-party software must be used to interpret this information to make security decisions. Typically, the solutions involve both client-side code and server/cloud-side software and services. On the client side, antivirus software will be enhanced to use attestation to report system configuration. On the server side, system health-monitoring applications or network-access control services will query the health of clients that request access. The server systems then receive and interpret attestation reports, and will grant network access (or raise an alert) based on the health reports received.

This paper is distributed with sample code and a utility called PCPTool. The tool and the provided sources are designed to help AV-system vendors make use of the attestation facilities in Windows.

Scope

The scope of this paper is an introduction to the Windows 8 capabilities, application programming interfaces (APIs), and properties around TPM 1.2 and 2.0 and a compilation of information required to design a solution. On a more practical level, the PCP-Kit is provided to allow direct experimentation and evaluation on a live Windows 8 operating system.

PCP-Kit and TPM Versions

At the time of writing, most TPMs are based on the TPM 1.2 specification published by the Trusted Computing Group (TCG). The new version of the TPM is referred to by Microsoft as "TPM 2.0"; however, the TCG refers to it as "TPM, family 2.0." The TPM 1.2 and TPM 2.0 devices are not compatible; however, Windows 8 supports both classes of devices, and some TPM-based services that are built into Windows 8 use an adaptation layer that transparently uses whatever device is present.

Windows supports attestation by allowing applications direct access to the underlying TPM so that the applications themselves can provide attestation services. This places the onus on the application to understand two complex devices. To lessen that burden, the PCP-Kit contains sample library code that provides a common hardware abstraction layer, as well as raising the functional level of the platform services provided.

We strongly suggest that solution providers obtain third-party TPM-library solutions or use or adapt the PCP-Kit library code to minimize application complexity.

Key Concepts Used in This Paper

Attestation is a complex subject. Solution providers can avoid some of the complexity if they use the code and utilities in the PCP-Kit, but some concepts and jargon must be explained so they can understand the facilities described in this paper. This section sketches some core TPM concepts, and provides a high-level overview of the Windows OS and PCP-Kit capabilities.

The Endorsement Key or EK

The TPM has an embedded unique cryptographic key called the *Endorsement Key*, or *EK*. The EK may be created during manufacturing, and for most TPMs the EK will remain in the TPM for the life of the device.

The EK is designed to provide a reliable cryptographic identifier for the platform. An enterprise might maintain a database of the Endorsement Keys belonging to the TPMs of all of the PCs in their enterprise, or a data center fabric controller might have a database of the TPMs in all of the blades. On Windows you can use the NCrypt provider described in the section "Platform Crypto Provider in Windows 8" to read the public part of the EK.

All TPMs compatible with Windows use a 2,048-bit RSA Endorsement Key. This TPM Endorsement Key is often accompanied by one or two *digital certificates.* One certificate is produced by the TPM manufacturer and is called the *endorsement certificate.* Another is produced by the platform builder and is called the *platform certificate* to indicate that a particular TPM is integrated with a certain machine. The owner of the machine may produce a third EK certificate to indicate ownership of the device. Such a certificate would be an *enterprise certificate*. On Windows you can use the NCrypt provider described in

the section “Platform Crypto Provider in Windows 8” to access the EK certificate store that contains all certificates for the EK on the platform. The administrator may add or remove certificates from this store.

The Endorsement Key can be used for direct machine authentication using the TPM functions TPM_ActivateIdentity (TPM 1.2) or TPM2_ActivateCredential (TPM 2.0). These functions are based on public key decryption using the Endorsement Key. However, the TPM allows secondary keys to be created that serve as delegates for the EK. These delegates are separate keys called *Attestation Identity Keys* or *AIKs*, and are described in the next section. Individual AIKs can be used with different services to avoid correlation based on the unique EK.

Attestation Identity Keys (AIKs)

Attestation Identity Keys are delegates for the EK that may be cryptographically certified by a third party called an *identity service provider* or *ISP*. Windows creates AIKs in the TPM that are 2,048-bit RSA signing keys. It is the job of the identity service provider to cryptographically establish that it is communicating with a real TPM (or a particular TPM, or a TPM in a group) and that the TPM has created a new AIK. Once the identity service provider has established these facts, it can create a new certificate for the new AIK-based identity.

The AIK certificate can contain whatever information the ISP thinks is appropriate. If the AIK is strictly to be used inside an enterprise it might be sufficient to simply record the public parts of the AIK in a database and no certificate for it is required. At the other end of the spectrum, the digital certificate might only record that the AIK belongs to a legitimate TPM. This might be appropriate for peer-to-peer trust on the Internet.

The key benefit of an AIK is that the server-side component to an attestation solution can verify that the measurements actually came from the specific client it intends to verify. This is why each server-side solution looking at data from a client system may want to establish its own AIK for a client, or use an AIK provisioned by an identity service provider it trusts to establish a trust association with the client to be verified.

The PCP-Kit contains sample code and tools for both client- and server-side AIK creation and certification. These functions are described more in the sections “Platform Crypto Provider in Windows 8” and “Attestation API Reference Implementation.”

The AIKs directly support attestation because they can be used to sign the platform configuration. The way that Windows 8 records platform configurations is described in the next section.

It is a best practice from a privacy perspective for solutions to leverage an identity service provider because it can provide a level of anonymity for clients.

Platform Configuration and Platform Configuration Registers (or PCRs)

The TPM contains a set of registers that are designed to provide a cryptographic representation of the software and state of the system that booted. These registers are called *platform configuration registers* or *PCRs*. PCRs are set to zero when the platform is booted, and is the job of the software that boots the platform to measure software in the boot chain and to record the measurements in the PCRs. Typically, boot components take the hash of the next component that is to be run and record the hash measurements in the PCRs. The initial component that starts the measurement chain is implicitly trusted. It is called the

Core Root of Trust for Measurement. Platform manufacturers are required to have a secure update process for the Core Root of Trust for Measurement or not permit updates to it.

The PCRs record a cumulative hash of the components that have been measured. The value in a PCR on its own is hard to interpret (it is just a hash value), but platforms typically keep a log with details of the software and configurations that have been recorded, and the PCRs merely ensure that the log has not been tampered with. The logs are described in more detail in the section "Windows Boot Configuration Log" and also in the TCG specifications[1].

In Windows, the OS boot components record the OS loader, the OS kernel and all boot-start drivers, and specially signed Early Launch Anti-Malware drivers, as well as any necessary configuration files. This means that PCRs can report both the precise details of the OS that is running, the precise ELAM driver that has been loaded and initialized, and the policy that is being checked or enforced by the ELAM driver (for instance, a hash value that represents a dated virus definition file). The ELAM driver is a small driver with a small policy database that has a very narrow scope, focused on drivers that are loaded early at system launch. The policy database is stored in a new registry hive that is also measured to the TPM, to record the operational parameters of the ELAM driver.

The ELAM driver is initialized by the OS and is responsible for ensuring that later-loading components and configurations are within its policy until the regular AV driver is loaded and initialized. If the ELAM driver detects a policy violation (a known rootkit, for example), it may invalidate the PCRs that indicated the system was in a good state. This is done with a new OS call named Tbsi_Revoke_Attestation(), and is described in more detail in the section "Invalidating the System Trust State." After the regular full-scale AV driver is initialized and running, the ELAM driver and the ELAM hive will be unloaded.

This relatively simple model is made somewhat more complex by system hibernate and resume cycles. This is described in more detail in the section "Platform Trust Considerations across Hibernation and Resume."

Attestation

This section describes how PCRs (that contain system configuration data) and AIKs (that can report platform state) are used for configuration reporting.

As already described, the platform firmware and the operating system – in conjunction with the ELAM driver – will ensure that the platform configuration registers and the associated TCG logs are an accurate representation of the platform state.

Before the platform can *report* its configuration using the TPM attestation functions, an AIK must be created or provisioned in conjunction with a third party to achieve strong trust in the key. Clients and servers can use the PCPTool command-line utility or the PCP-Kit TpmAttPubKeyFromIdBinding and TpmAttGenerateActivation library functions to perform these actions (as described in the section "Creating Attestation Identity Keys (AIKs) and Forming Remote Trust").

Once provisioned, the AIK can be used in conjunction with the PCP-Kit sample/library code routines such as TpmAttCreateAttestationfromLog to report platform configuration. If the AIK is placed at a defined location in the registry, the OS will also create a signature over the platform log state (and a monotonic counter value) at each boot.

[1] Please refer to www.trustedcomputinggroup.org/developers/pc_client

Typically, the logs and statements of platform health are interpreted on servers. Checking that a TPM attestation and the associated log are valid takes several steps.

First, the server must check that the reports are signed by trustworthy AIKs. This might be done by checking that the public part of the AIK is listed in a database of assets, or perhaps that a certificate has been checked.

Once the key has been checked, the signed attestation (a quote structure) should be checked to see whether it is a valid signature over PCR values. Server code can use the TpmAttValidateKeyAttestation library routine, or the PCPTool utility.

Next the logs should be checked to ensure that they match the PCR values reported.

Finally, the logs themselves should be examined to see whether they represent known or valid security configurations. For instance, a simple check might be to see whether the measured early OS components are known good, that the ELAM driver is as expected, and that the ELAM-driver policy file is up-to-date (these checks are beyond the scope of this paper).

If all of these checks succeed, an attestation statement can be issued that later can be used to determine whether or not the client should be granted access to a resource.

Acronyms and Abbreviations

ACPI – Advanced Configuration and Power Interface

AIK – Attestation Identity Key. A TPM key that serves as an identity for the computer platform.

BCrypt – Algorithm provider infrastructure in CNG

CA – Certificate Authority

CAPI – Crypto API in Windows

Cert – Short for certificate

CNG – Crypto Next Generation API in Windows. This API is the successor of Crypto API (CAPI).

EK – Endorsement Key. A TPM key that is a cryptographic identifier for the TPM.

ELAM – Early Launch Anti-Malware. A driver that is loaded early by Windows and is responsible for checking and enforcing the early boot security policy.

EPS – Endorsement Primary Seed. A value from which a TPM 2.0 Endorsement Key is generated.

Key_{PUB} – Public portion of an asymmetric key

KSP – Key Storage Provider, a service of CNG that is accessed with the NCrypt APIs

NCrypt – Key storage provider infrastructure in CNG

PCP – Platform Crypto Provider

PCPKSP – Platform Crypto Provider Key Storage Provider

PPS – Platform Primary Seed. A value from which a TPM 2.0 platform key is generated.

SPS – Storage Primary Seed. A value from which a TPM 2.0 SRK is generated.

SRK – Storage Root Key

TBS – TPM Base Services

TCG – The Trusted Computing Group. The organization that is responsible for the TPM specification and related standards.

TPM – Trusted Platform Module

WBCL – Windows Boot Configuration Log

VSC – virtual smart card

TPM Provisioning and Management

The Windows 8 operating system is designed to automatically provision the TPM. This is in contrast to earlier versions of Windows where the end user had to provide explicit administrative actions. Once provisioned, Windows retains enough information to enable advanced TPM scenarios for itself or third-party applications. This also contrasts with earlier Windows versions where an application or TCG Software Stack (TSS) needed the TPM owner authorization value to be explicitly specified in order to perform advanced TPM scenarios.

Pre-Windows 8 Architecture for TPM 1.2 Provisioning

Most TPM systems in the marketplace in 2011 ship with the TPM 1.2 *deactivated* and *disabled*. When the TPM is deactivated it cannot be used for any interesting scenarios. We call the act of making the TPM ready to use *provisioning.* On current systems, the computer-platform firmware typically requires someone physically present (sitting at the keyboard) to approve changing the TPM state during the boot process.

This architecture has proven very complex for owners and system administrators. Few end users will understand unexpected BIOS/firmware prompts and questions, and system administrators dislike the fact that they cannot really do remote administration.

Some OEMs have provided tools and utilities to simplify remote administration, but these are hard to deploy in heterogeneous environments.

Windows 8 drastically simplifies the provisioning of TPM systems, as described in the next few sections.

Auto-Provisioning

Windows 8 will auto-provision the TPM so it is ready for use when applications want to use it. Windows 8 will not auto-provision the TPM if the provisioning process requires administrator interaction to complete the process.

Auto-provisioning actions will happen shortly after a full boot completes on a system with Windows 8. This means auto-provisioning usually won't occur after resuming from sleep or hibernation, including when hibernation technology hibernates the system core. (Generally, choosing to restart the system will cause a full boot.)

If the auto-provisioning actions find the TPM is deactivated or disabled, the provisioning actions will check the system capabilities to determine if it implements the TCG Physical Presence Interface Specification 1.2 and then if the system has the NoPPIProvision flag set to TRUE. If the flag is TRUE, it means the OS can initiate enabling and activating the TPM without any user involvement. The OS will request that the firmware enable and activate the TPM on the next boot. Upon the next restart of the OS, the TPM will be enabled and activated.

If the auto-provisioning actions find the TPM is enabled, activated, and ready to have ownership taken, the OS will take ownership of the TPM and perform a series of actions to set it up for use.

Scenarios that could cause Windows to not complete auto-provisioning are:

- The TPM is disabled or deactivated and the system does not implement the TCG Physical Presence Interface Specification 1.2 or does not have the NoPPIProvision flag set to TRUE.
- Ownership of the TPM was previously taken. This can happen if Windows is erased and reinstalled without clearing the TPM. It can also happen with an upgrade from a machine running Windows 7 or Windows Vista® that is using BitLocker® Drive Encryption.
- Additional Windows actions associated with provisioning the TPM cannot be completed. An example would be if Group Policy is configured to back up the TPM owner authorization value to Active Directory® service, but no connection to Active Directory is available.

Provisioning Through the UI

To manually provision the TPM, an administrator or an application can use the TPM Provisioning Wizard. The wizard is named tpminit.exe and may be run as an executable (Start->Run->tpminit.exe). The wizard will interactively walk the administrator through the process of provisioning the TPM.

TPM State in the OS

The TPM state in the Windows 8 operating system has been simplified for the user interface. The TPM may show one of the following statuses:

- "Ready for use" – The TPM and system are fully provisioned for TPM-related uses.
- "Ready for use, with Reduced Functionality" – Ownership of the TPM has been taken, but other system or TPM configuration actions are not completed.
- "Not Ready" – The TPM and system are not ready for use.

Provisioning with WMI

Several new Windows 8 Windows Management Instrumentation (WMI) methods help with TPM management. The methods IsReady and IsReadyInformation are useful for determining whether the TPM is fully ready for use or why it isn't. Another method, Provision, encapsulates the complex logic Windows 8 uses to provision the TPM into a single method for third- parties to use if they want to present their own provisioning user interface.

IsReady

This method determines whether the TPM and system are fully provisioned and ready for TPM use. The TPM may still be useful for some scenarios even if this method returns FALSE.

```
uint32 IsReady(

      [OUT] boolean IsReady

);
```

Parameters:

`[OUT] boolean IsReady` – Set to TRUE if the TPM and system are fully provisioned for TPM use.

Return value:

If the function succeeds, the function returns S_OK (0).

Remarks:

The return value indicates more than just TPM state. For example, if the system is configured to back up the TPM owner authorization value to Active Directory and the task has not been completed, IsReady will be set to FALSE.

This method is expensive to run because it performs many checks and does not cache its return values. It is recommended applications use this method only when necessary (for example, for provisioning and for troubleshooting).

IsReadyInformation

This method returns the status of the TPM and system and whether or not the TPM is provisioned and ready for use.

```
uint32 IsReadyInformation(

      [OUT] boolean IsReady,

      [OUT] uint32 Information

);
```

Parameters:

`[OUT] boolean IsReady` – Set to TRUE if the TPM and system are fully provisioned for TPM use.

`[OUT] uint32 Information` – Returns a bitmask of as much information as is available of what is needed to fully provision the TPM.

The Information bitmask may consist of the following values:

Symbol	Value	Description
INFORMATION_SHUTDOWN	0x00000002	Platform restart is required (shutdown).
INFORMATION_REBOOT	0x00000004	Platform restart is required (reboot).
INFORMATION_TPM_FORCE_CLEAR	0x00000008	The TPM is already owned. Either the TPM needs to be cleared or the TPM owner authorization value needs to be imported.
INFORMATION_PHYSICAL_PRESENCE	0x00000010	Physical Presence is required to provision the TPM.
INFORMATION_TPM_ACTIVATE	0x00000020	The TPM is disabled or deactivated.
INFORMATION_TPM_TAKE_OWNERSHIP	0x00000040	Ownership was taken.
INFORMATION_TPM_CREATE_EK	0x00000080	An Endorsement Key was created.
INFORMATION_TPM_OWNERAUTH	0x00000100	The TPM owner authorization is not properly stored in the registry.
INFORMATION_TPM_SRK_AUTH	0x00000200	The Storage Root Key authorization value is not all zeros.
INFORMATION_TPM_DISABLE_OWNER_CLEAR	0x00000400	Windows is configured to disable clearing of the TPM with the TPM owner authorization value and the TPM has not been configured to prevent the clearing.
INFORMATION_TPM_SRKPUB	0x00000800	The Windows registry information about the TPM's Storage Root Key does not match the

		TPM Storage Root Key.
INFORMATION_TPM_READ_SRKPUB	0x00001000	The TPM permanent flag to allow reading of the Storage Root Key public value is not set.
INFORMATION_TPM_BOOT_COUNTER	0x00002000	The monotonic counter incremented during boot has not been created.
INFORMATION_TPM_AD_BACKUP	0x00004000	The TPM's owner authorization has not been backed up to Active Directory.
INFORMATION_TPM_AD_BACKUP_PHASE_I	0x00008000	The first portion of the TPM owner authorization information storage in Active Directory is in progress.
INFORMATION_TPM_AD_BACKUP_PHASE_II	0x00010000	The second portion of the TPM owner authorization information storage in Active Directory is in progress.
INFORMATION_LEGACY_CONFIGURATION	0x00020000	Windows Group Policy is configured to not store any TPM owner authorization so the TPM cannot be fully ready.
TBD	0x00040000	The EK certificate was not read from the TPM non-volatile (NV) RAM and stored in the registry. (See the NCrypt Property NCRYPT_PCP_EKCERT_PROPERTY for more information.)
INFORMATION_TCG_EVENT_LOG	0x00080000	The TCG event log is empty or cannot be read. (This is usually a problem with the system firmware, not TPM state.)
INFORMATION_NOT_REDUCED	0x00100000	The TPM is not owned or otherwise not ready for use by BitLocker.
INFORMATION_GENERIC_ERROR	0x00200000	A generic error occurred.

Return value:
If the function succeeds, the function returns S_OK (0).

Remarks:
This method is expensive to run because it performs many checks and does not cache its return values. It is recommended applications use this method only when necessary (for example, for provisioning and for troubleshooting).

Provision

This method provisions the TPM and the system for use.

```
uint32 Provision(
      [IN] boolean ForceClear_Allowed,
      [IN] Boolean PhysicalPresencePrompts_Allowed,
      [OUT] uint32 Information
);
```

Parameters:

`[IN] boolean ForceClear_Allowed` – When set to TRUE, the method may request Physical Presence operations to clear the TPM. If set to FALSE, the method will not request a Physical Presence operation to clear the TPM.

`[IN] Boolean PhysicalPresencePrompts_Allowed` – When set to TRUE, the method may request Physical Presence operations that require user involvement during the boot process to confirm the TPM state change.

`[OUT] uint32 Information` – Returns a bitmask of as much information as is available of what is needed to fully provision the TPM. Mask values like INFORMATION_REBOOT indicate the method call should initiate a reboot to move the provisioning process forward.

The Information bitmask may consist of the following values:

Symbol	Value	Description
INFORMATION_SHUTDOWN	0x00000002	Platform restart is required (shutdown).
INFORMATION_REBOOT	0x00000004	Platform restart is required (reboot).
INFORMATION_TPM_FORCE_CLEAR	0x00000008	The TPM is already owned. Either the TPM needs to be cleared or the TPM owner authorization value needs to be imported.
INFORMATION_PHYSICAL_PRESENCE	0x00000010	Physical Presence is required to provision the TPM.
INFORMATION_TPM_ACTIVATE	0x00000020	The TPM is disabled or deactivated.
INFORMATION_TPM_TAKE_OWNERSHIP	0x00000040	Ownership was taken.
INFORMATION_TPM_CREATE_EK	0x00000080	An Endorsement Key was created.
INFORMATION_TPM_OWNERAUTH	0x00000100	The TPM owner authorization is not properly stored in the registry.
INFORMATION_TPM_SRK_AUTH	0x00000200	The Storage Root Key authorization value is not all zeros.
INFORMATION_TPM_DISABLE_OWNER_CLEAR	0x00000400	Windows is configured to disable clearing of the TPM with the TPM owner authorization value and the TPM has not been configured to prevent the clearing.
INFORMATION_TPM_SRKPUB	0x00000800	The Windows registry information about the TPM's Storage Root Key does not match the TPM Storage Root Key.
INFORMATION_TPM_READ_SRKPUB	0x00001000	The TPM permanent flag to allow reading of the Storage Root Key public value is not set.
INFORMATION_TPM_BOOT_COUNTER	0x00002000	The monotonic counter incremented during boot has not been created.
INFORMATION_TPM_AD_BACKUP	0x00004000	The TPM's owner authorization has not been backed up to Active Directory.
INFORMATION_TPM_AD_BACKUP_PHASE_I	0x00008000	The first portion of the TPM owner authorization information storage in Active Directory is in progress.
INFORMATION_TPM_AD_BACKUP_PHASE_II	0x00010000	The second portion of the TPM owner authorization information storage in Active Directory is in progress.

INFORMATION_LEGACY_CONFIGURATION	0x00020000	Windows Group Policy is configured to not store any TPM owner authorization so the TPM cannot be fully ready.
TBD	0x00040000	The EK certificate was not read from the TPM NV RAM and stored in the registry. (See the NCrypt Property NCRYPT_PCP_EKCERT_PROPERTY for more information.)
Symbol	Value	Description
INFORMATION_TCG_EVENT_LOG	0x00080000	The TCG event log is empty or cannot be read. (This is usually a problem with the system firmware, not TPM state.)
INFORMATION_NOT_REDUCED	0x00100000	The TPM is not owned or otherwise not ready for use by BitLocker.
INFORMATION_GENERIC_ERROR	0x00200000	A generic error occurred.

Return value:
If the function succeeds, the function returns S_OK (0).

Remarks:
This method is expensive to run because it performs many checks. It is recommended applications use this method only when necessary.

Provisioning Differences Between TPM Versions 1.2 and 2.0

In terms of provisioning, TPM 1.2 and TPM 2.0 have a variety of differences regarding TPM state and TPM management authorization values. Windows 8 has roughly equivalent actions it takes for both versions of TPM to provision either. Both are auto-provisioned, both use the TPM Management Console and both are able to be manipulated through the WMI Win32_TPM class.

Windows 8 Certified Hardware Requirements

Microsoft has a Windows 8 hardware certification program that helps OEMs ship systems with the best defaults and capabilities for a great customer experience. Windows 8 systems with a TPM should have the following characteristics:

- Implement the TCG Physical Presence Interface Specification 1.2, setting the NoPPIProvision flag to TRUE by default. When ownership of the TPM has not been taken, this flag's setting allows the OS to ready the TPM for use without a physically present user being involved in the provisioning process.
- The TPM will be enumerated to Windows by default. This means Windows sees the Advanced Configuration and Power Interface (ACPI) device object for the TPM and can manage it.
- A full Endorsement Key Certificate provisioned in the TPM's NV RAM using the TPM_NV_INDEX_EKCert index location described in section 4.2.1 of the TCG PC Client Specific Implementation Specification for Conventional BIOS using the structures described in section 7.4 if the TPM does not support generating a new Endorsement Key.

Platform Crypto Provider in Windows 8

Support for the TPM in Windows 8 has been significantly expanded. Crypto-operations is one such area of expanded support. The TPM can now be used for crypto-operations through the standard Crypto Next Generation (CNG) Windows interfaces.

The new CNG-Platform Crypto Provider can use a TPM 1.2 or 2.0 device to provide TPM version-independent crypto services. The provider may be used as a replacement for the Microsoft software provider because it provides a superset of its properties (with the exception of exportable keys that are supported only in an authorized form).

The benefit of using the provider instead of issuing TPM commands directly through the *TPM Based Services* (TBS) is that the provider abstracts TPM version differences and takes care of TPM key management for the application. The provider allows an application to switch between TPM or software based keys by changing to a different provider name. This allows an application provider to keep TPM-specific code to a minimum, meaning the application will also run on machines with no TPM or in cases where the user prefers to not use the TPM.

The Platform Crypto Provider should not be set as default RSA provider on the system because it does not operate with the same performance that the Microsoft software provider does. Also, the software provider supports unauthorized exportable keys that do not make sense for the Platform Crypto Provider.

Certificate Enrollment with the Platform Crypto Provider

To demonstrate certificate enrollment with the TPM-based Platform Crypto Provider, a self-signed certificate can be created with the certreq.exe utility as follows: '`certreq.exe –new PCPCert.inf PCPCert.cer`' where the file PCPCert.inf has the contents:

```
[NewRequest]
   Subject = "CN=SelfSignedPCPCert"
   HashAlgorithm = sha256
   KeyAlgorithm = RSA
   KeyLength = 2048
   KeyUsage = "CERT_DIGITAL_SIGNATURE_KEY_USAGE"
   KeyUsageProperty = "NCRYPT_ALLOW_SIGNING_FLAG"
   ProviderName = "Microsoft Platform Crypto Provider"
   RequestType = Cert
   FriendlyName = "DeleteMe!"
   Exportable = false
[EnhancedKeyUsageExtension]
   OID=2.5.29.37.0
```

In order to use the provider in an enterprise scenario with a Windows 8 Certificate Authority (CA) and Policy Server, the server has to have a TPM that is *ready* and may be using the TPM to protect the CA signing keys, for example. The CA administrator then creates a certificate template that mandates the usage of the Platform Crypto Provider for key storage on the client. Clients that enroll for this certificate will make use of the TPM without any action from their side.

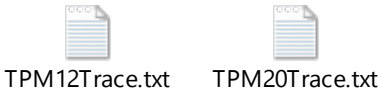
Tracing Provider TPM Commands

Advanced users who have applications that are issuing their own TPM commands or are debugging other complex issues can audit the operations that the provider executes on the TPM and can turn on provider tracing to receive a decoded trace of the TPM communication by setting a REG_SZ value `ProviderTraces` under the key

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TPM
```

with a folder name. If set, the provider will write TPM trace log files in this directory. The trace log files are identified by the process ID. (The application using the Platform Crypto Provider must have write access to the directory because the provider executes in-process).

Trace examples of a key creation:



Administrator privileges are required to set the registry value because the trace files may contain secrets that are passed to and from the TPM. Note also that this feature will produce substantial amounts of data on the disk.

BCrypt RNG Platform Crypto Provider

The BCrypt Platform Crypto Provider exposes the TPM random number generator (RNG) directly through the CNG RNG interface. This is in contrast to the default Windows RNG provider, which uses many sources of entropy (the TPM is one source).

Supported Functions

To open the provider, call BCryptOpenAlgorithmProvider with *pszAlgId* = BCRYPT_RNG_ALGORITHM and specify the provider with *pszImplementation* = MS_PLATFORM_CRYPTO_PROVIDER.

A call to BCryptGenRandom will fill the caller-provided buffer with random data. The maximum random number size is limited to 4,096 bytes per call. The provider may make several round trips to the TPM to fill the caller's buffer and the call will block until the request is satisfied.

The provider also supports "stirring" the RNG in the TPM, by providing the flag BCRYPT_RNG_USE_ENTROPY_IN_BUFFER with the call BCryptGenRandom. If this flag is set, the provider will stir the entropy generator in the TPM with the data in the caller's buffer. If the buffer is larger than 256 bytes, only the first 256 bytes will be used.

Supported Properties

BCRYPT_ALGORITHM_NAME	Algorithm name as defined in MSDN	GET
BCRYPT_PCP_PLATFORM_TYPE_PROPERTY	String that identifies the TPM manufacturer and version	GET
BCRYPT_PCP_PROVIDER_VERSION_PROPERTY	Crypto Provider Version	GET

NCrypt RSA Platform Key Storage Provider

The Platform Crypto Provider's Key Storage Provider (KSP) provides a subset of the functionality that is provided by the Microsoft software provider. It supports all functionality that is required by certificate enrollment. The Platform Crypto Provider does not support regular exportable keys, because those are by definition not bound to the platform and there would be no security benefit by using the TPM for these keys. However, the provider does support authenticated exportable keys that use the *migrationAuth* for a controlled export. The *migrationAuth* is a Platform Crypto Provider-specific key property that allows the TPM to enforce key export control of key material.

Beyond the well-defined provider and key properties, the Platform Crypto Provider supports a number of special properties that enable specific TPM-related scenarios.

Supported Functions

- NCryptCreatePersistedKey
- NCryptDecrypt
- NCryptDeleteKey
- NCryptEncrypt
- NCryptEnumAlgorithms
- NCryptEnumKeys
- NCryptExportKey
- NCryptFinalizeKey
- NCryptFreeBuffer
- NCryptFreeObject
- NCryptGetProperty
- NCryptImportKey
- NCryptIsAlgSupported
- NCryptOpenKey
- NCryptOpenStorageProvider
- NCryptSetProperty
- NCryptSignHash
- NCryptVerifySignature

Supported Properties

Properties that are exclusively provided by the PCP provider are marked in gray.

Provider Properties		
BCRYPT_ALGORITHM_NAME	Cryptography Primitive Property Identifiers as defined in MSDN	GET
BCRYPT_PADDING_SCHEMES	Cryptography Primitive Property Identifiers as defined in MSDN	GET
BCRYPT_KEY_LENGTHS	Cryptography Primitive Property Identifiers as defined in MSDN	GET
NCRYPT_USE_CONTEXT_PROPERTY	Key Storage Property Identifiers as defined in MSDN	SET
NCRYPT_PCP_PLATFORM_TYPE_PROPERTY	String that identifies the TPM manufacturer and version	GET
NCRYPT_PCP_PROVIDER_VERSION_PROPERTY	Crypto Provider Version	GET
NCRYPT_PCP_EKPUB_PROPERTY	EK_{PUB} from the TPM as a BCRYPT_RSAKEY_BLOB structure	GET
NCRYPT_PCP_EKCERT_PROPERTY	Read/Write HCERTSTORE handle to the EKCert store in the registry. The caller may persist multiple custom-generated EK certs in the store and may delete the TPM manufacturer-provided one if so desired. The TPM manufacturer-provided EK cert may always be read from the property NCRYPT_PCP_EKNVCERT_PROPERTY.	GET
NCRYPT_PCP_EKNVCERT_PROPERTY	EKCert directly from the TPM NVRAM (if present)	GET
NCRYPT_PCP_SRKPUB_PROPERTY	SRK_{PUB} from the TPM as BCRYPT_RSAKEY_BLOB structure	GET
NCRYPT_PCP_PCRTABLE_PROPERTY	All 24 SHA-1 PCRs in ascending order from PCR[0] to PCR[23] returned as a packed array of binary arrays	GET
NCRYPT_PCP_PLATFORMHANDLE_PROPERTY	The TBS handle that the provider uses to submit commands to the TPM. A provider platform key handle is only valid within this TBS context.	GET
NCRYPT_PCP_ALTERNATE_KEY_STORAGE_LOCATION_PROPERTY	Specify an alternate base key storage path for the provider operation	GET

NCRYPT_PROVIDER_HANDLE_PROPERTY	Key Storage Property Identifiers as defined in MSDN	GET
NCRYPT_NAME_PROPERTY	Key Storage Property Identifiers as defined in MSDN	GET
NCRYPT_UNIQUE_NAME_PROPERTY	Key Storage Property Identifiers as defined in MSDN	GET
NCRYPT_MAX_NAME_LENGTH_PROPERTY	Key Storage Property Identifiers as defined in MSDN	GET
NCRYPT_WINDOW_HANDLE_PROPERTY	Key Storage Property Identifiers as defined in MSDN	GET; SET
NCRYPT_USE_CONTEXT_PROPERTY	Key Storage Property Identifiers as defined in MSDN	GET; SET
NCRYPT_KEY_USAGE_PROPERTY	Key Storage Property Identifiers as defined in MSDN	GET; SET
NCRYPT_EXPORT_POLICY_PROPERTY	Key Storage Property Identifiers as defined in MSDN For SET, only NCRYPT_ALLOW_ARCHIVING_FLAG with NCRYPT_ALLOW_PLAINTEXT_ARCHIVING_FLAG is supported. GET will return NCRYPT_ALLOW_EXPORT_FLAG with NCRYPT_ALLOW_PLAINTEXT_EXPORT_FLAG if a key may be exported with an export password.	GET; SET
NCRYPT_UI_POLICY_PROPERTY	Key Storage Property Identifiers as defined in MSDN	GET; SET
NCRYPT_KEY_TYPE_PROPERTY	Key Storage Property Identifiers as defined in MSDN	GET
NCRYPT_LENGTH_PROPERTY	Key Storage Property Identifiers as defined in MSDN Only 2,048- (default) and 1,024-bit keys are supported.	GET; SET

NCRYPT_LENGTHS_PROPERTY	Key Storage Property Identifiers as defined in MSDN	GET
NCRYPT_ALGORITHM_PROPERTY	Key Storage Property Identifiers as defined in MSDN	GET
NCRYPT_ALGORITHM_GROUP_PROPERTY	Key Storage Property Identifiers as defined in MSDN	GET
NCRYPT_CERTIFICATE_PROPERTY	Key Storage Property Identifiers as defined in MSDN	GET; SET
NCRYPT_LAST_MODIFIED_PROPERTY	Key Storage Property Identifiers as defined in MSDN	GET
NCRYPT_SECURITY_DESCR_PROPERTY	Key Storage Property Identifiers as defined in MSDN	GET; SET
NCRYPT_PIN_PROPERTY, BCRYPT_PCP_PASSWORD_PROPERTY	Key Storage Property Identifiers as defined in MSDN	SET
NCRYPT_PCP_MIGRATIONPASSWORD_PROPERTY	Migration password for a key. Setting this property before it is finalized will make a key exportable.	SET
NCRYPT_PCP_PLATFORM_TYPE_PROPERTY	String that identifies the TPM manufacturer and version	GET
NCRYPT_PCP_PROVIDER_VERSION_PROPERTY	Crypto Provider Version	GET
NCRYPT_PCP_STORAGEPARENT_PROPERTY	Optional handle to a storage key that this key is protected by. Has to be set before the key is finalized and every time the key is loaded.	GET; SET
BCRYPT_KEY_LENGTH, BCRYPT_KEY_STRENGTH	Cryptography Primitive Property Identifiers as defined in MSDN	GET; SET
BCRYPT_BLOCK_LENGTH, BCRYPT_SIGNATURE_LENGTH	Cryptography Primitive Property Identifiers as defined in MSDN	GET

NCRYPT_PCP_PLATFORMHANDLE_PROPERTY	Virtualized Key handle that this key uses in the TPM. This property in conjunction with the provider NCRYPT_PCP_PLATFORMHANDLE_PROPERTY may be used to issue custom commands on a loaded key. The handle is only valid within the TBS context that the provider uses.	GET
NCRYPT_PCP_KEY_USAGE_POLICY	TPM key usage restriction and special TPM keys: NCRYPT_TPM12_PROVIDER = (0x00010000) NCRYPT_PCP_SIGNATURE_KEY = (0x00000001) NCRYPT_PCP_ENCRYPTION_KEY = (0x00000002) NCRYPT_PCP_GENERIC_KEY = (NCRYPT_PCP_SIGNATURE_KEY | NCRYPT_PCP_ENCRYPTION_KEY) NCRYPT_PCP_STORAGE_KEY = (0x00000004) NCRYPT_PCP_IDENTITY_KEY = (0x00000008)	GET; SET
NCRYPT_PCP_PASSWORD_REQUIRED_PROPERTY	BOOLEAN value that can be queried to discover if a key requires authentication prior to use	GET
NCRYPT_PCP_EXPORT_ALLOWED_PROPERTY	BOOLEAN value that can be queried to discover if a key is exportable. This value will also return TRUE if a key was generated outside the TPM to indicate that the key material was at least at some point in existence outside the TPM. For key import on TPM 1.2, if no explicit export password was set, a randomly generated unknown value is set for the export password.	GET
NCRYPT_PCP_USAGEAUTH_PROPERTY	TPM authValue for the key (typically the SHA-1 digest of the usage password)	GET; SET
NCRYPT_PCP_TPM12_IDBINDING	For keys with usage policy NCRYPT_PCP_IDENTITY_KEY only. SET will set the nonce from the CA before the key is finalized and after the key is finalized the ID Binding may be retrieved with GET. Only valid on initial key handle. This property is not persisted. The IDBinding returned by this property consists of a structure containing the public portion of the AIK	GET; SET

	along with other data, and a signature over this structure.	
NCRYPT_PCP_TPM12_IDACTIVATION	For keys with usage policy NCRYPT_PCP_IDENTITY_KEY only. SET will set the challenge from the CA and the EK wrapped secret may be retrieved with GET. Administrative privileges are required.	GET; SET
NCRYPT_PCP_PLATFORM_BINDING_PCRMASK_PROPERTY	Before the key is finalized, SET allows a UINT32 mask value to be provided that binds key usage to the indicated set of PCR values. Bit 0 of the mask corresponds with PCR[0] and bit 23 with PCR[23]. If no PCR table is provided, the current PCR values in the TPM are used. GET will return the mask if a key was bound to PCRs.	GET; SET
NCRYPT_PCP_PLATFORM_BINDING_PCRDIGESTLIST_PROPERTY	Provide all 24 SHA-1 PCRs that are to be used for PCR binding during key creation as a packed binary array, with PCR[0] first	SET
NCRYPT_PCP_PLATFORM_BINDING_PCRDIGEST_PROPERTY	Return the PCR digest a key is bound to. This is the SHA-1 digest of the TPM_PCR_COMPOSITE structure on TPM 1.2 and the SHA-256 digest of the PCRs to be provided to the TPM2_PolicyPCR command on TPM 2.0.	GET
NCRYPT_PCP_CHANGEPASSWORD_PROPERTY	Allows changing a usage password on a fully authorized key	SET
NCRYPT_PCP_KEYATTESTATION_PROPERTY	If at least one AIK is registered, the provider will generate key certification data for every non-exportable key that is created. This property allows access to this data after the key is finalized. The attestation data is stored persistent with the key.	GET

Description of Provider Data Structures

BCRYPT_OPAQUE_KEY_BLOB

The Platform Crypto Provider allows export and import of TPM-protected key blobs for backup and restore purposes or to use that key blob directly in a TPM command. This functionality is only available on the Platform Crypto Provider.

```
#define BCRYPT_PCP_KEY_MAGIC 'MPCP' // Platform Crypto Provider Magic
#define PCPTYPE_TPM12 (0x00000001)  // TPM type 1.2
#define PCPTYPE_TPM20 (0x00000002)  // TPM type 2.0
```

On TPM 1.2 systems, the key blob has a PCP_KEY_BLOB header followed by the indicated data in order:

```
typedef enum PCP_KEY_FLAGS {
    PCP_KEY_FLAGS_authRequired = 0x00000001   // Key uses authorization
} PCP_KEY_FLAGS;

typedef struct PCP_KEY_BLOB
{
    DWORD   magic;              // BCRYPT_PCP_KEY_MAGIC
    DWORD   cbHeader;           // Size of the header structure
    DWORD   pcpType;            // TPM type
    DWORD   flags;              // PCP_KEY_FLAGS Key flags
    ULONG   cbTpmKey;           // Size of TPM_KEY12 structure
} PCP_KEY_BLOB, *PPCP_KEY_BLOB;
```

On TPM 2.0 systems, the key blob has a PCP_KEY_BLOB_WIN8 header followed by the indicated data in order:

```
typedef enum PCP_KEY_FLAGS_WIN8 {
    PCP_KEY_FLAGS_WIN8_authRequired = 0x00000001  // Key uses authorization
} PCP_KEY_FLAGS_WIN8;

typedef struct PCP_KEY_BLOB_WIN8
{
    DWORD   magic;              // BCRYPT_PCP_KEY_MAGIC
    DWORD   cbHeader;           // Size of the header structure
    DWORD   pcpType;            // TPM type
    DWORD   flags;              // PCP_KEY_FLAGS_WIN8 Key flags
    ULONG   cbPublic;           // Size of Public key
    ULONG   cbPrivate;          // Size of Private key blob
    ULONG   cbMigrationPublic;  // Size of Public migration authorization object
    ULONG   cbMigrationPrivate; // Size of Private migration authorization object
    ULONG   cbPolicyDigestList; // Size of List of policy digest branches
    ULONG   cbPCRBinding;       // Size of PCR binding mask
    ULONG   cbPCRDigest;        // Size of PCR binding digest
    ULONG   cbEncryptedSecret;  // Size of hostage import symmetric key
    ULONG   cbTpm12HostageBlob; // Size of hostage import private key
} PCP_KEY_BLOB_WIN8, *PPCP_KEY_BLOB_WIN8;
```

NCRYPT_PCP_TPM12_IDBINDING

SET only accepts an SHA-1 digest on TPM 1.2 and 2.0 systems.

On TPM 1.2 systems, GET provides the following data after key finalize:

TPM_IDENTITY_CONTENTS || SIGNATURE

On TPM 2.0 systems, GET provides the following data after key finalize:

TPM2B_PUBLIC || TPM2B_CREATION_DATA || TPM2B_ATTEST || TPMT_SIGNATURE

Automatic Key Attestation

The attestation data that is returned from the property NCRYPT_PCP_KEYATTESTATION_PROPERTY is formatted as follows. It starts with a tag identifying the version, followed by attestation packages.

```
    ULONG    tag                // 'AK1T' for 1.2 and 'AK2T' for 2.0 TPM data
{
    ULONG    attestationSize    // There will be one attestation section for each
                                   registered Attestation Key
    USHORT   sizeAikName
    WCHAR    keyName[]          // Name under which the key is stored in the registry
    USHORT   sizeAikPubDigest
    BYTE     aikPubDigest[]     // SHA-1 Digest for the AIK modulus
    USHORT   sizeAttestationData
    BYTE     attestationData[]  // TPM-generated attestation structure
    USHORT   sizeSignature
    BYTE     signature[]       // AIK signature over the SHA-1 digest of attestation data
}
{
  // Second attestation package as above
}
```

NCRYPT_PCP_TPM12_IDACTIVATION

Use SET on TPM 1.2 systems to provide the EK_{PUB} encrypted structure TPM_EK_BLOB that contains the structure TPM_EK_BLOB_ACTIVATE.

Use SET on TPM 2.0 systems to set the activation credential:

TPM2B_ID_OBJECT || TPM2B_ENCRYPTED_SECRET

Use GET to perform the activation and retrieve the wrapped secret.

Using Storage Provider Keys with Custom Commands

If an application needs to run TPM commands that are not supported by the crypto provider, the provider can be queried for the relevant handles and contexts.

1. Open the crypto provider and create or open one or more keys to be used in a custom command.
2. Obtain the TBS handle hTbsContext from the provider with NCryptGetProperty(hProvider, NCRYPT_PCP_PLATFORMHANDLE_PROPERTY,…).
3. Obtain each key handle required for the custom command NCryptGetProperty($hKey_n$, NCRYPT_PCP_PLATFORMHANDLE_PROPERTY,…).
4. Get the TPM version on the platform from the TBS with the new function Tbsi_GetDeviceInfo().
5. Format the TPM-specific command (1.2 or 2.0 depending on what the previous call returned) structure using the obtained key handles.
6. Submit the custom command to the TPM with Tbsip_Submit_Command(hTbsContext, …).

Do not close the hTbsContext or the virtualized key handles in the TPM because this will put the provider in an undefined state. Closing the keys through the provider and closing the provider itself will release all resources.

Executing Custom TPM Commands Through the TBS API

TPM Base Services (TBS) is the Windows facility that allows commands to be sent to the underlying TPM without the abstraction provided by BCrypt. This section describes TBS functionality as well as auxiliary services such as access to the Windows boot configuration log (WBCL) and intentionally invalidating attestation.

TBS requires the calling application to prepare properly formatted TPM command buffers that are compatible with the underlying TPM (TPM 1.2 or TPM 2.0). However, Windows provides services that allow the TPM and its internal resources to be shared. These facilities are described in the next section.

TPM Resource Virtualization

One service provided by TBS is sending preformatted TPM commands to the TPM and providing the response back to the application. To isolate multiple applications from each other with respect to their effects on TPM state, TBS implements TPM context management. The TPM context management virtualizes certain limited TPM resources, and restricts access to these resources to the applications that created them.

To support this, TBS provides the notion of a *TPM context*, a resource container holding transient TPM resources that have been created by an application. Applications must first create such a TPM context, or use a previously created context such as that provided by the BCrypt provider. To run commands on the TPM, applications will then supply the TPM context together with the preformatted TPM command buffers.

TPM commands and responses contain *handles* for resources that are used or created. When an application sends a command to TBS to be run on the TPM, TBS examines the command. TBS ensures that transient resources referenced by the command have been created within the associated TPM context. On the way back, the TBS analyzes the TPM response to learn about newly created resources. If the limited resource slots of the TPM are filled, TBS will also transparently context-save some resources out of the TPM to make room for executing the current command. When these context-saved resources are referenced later by another TPM command, TBS will transparently reload these resources before executing the command (potentially context-saving other resources). This transparent context-saving and context-loading of resources is referred to as *resource management*.

Because TPM resource handles for transient objects change whenever these objects are reloaded into the TPM, TBS provides its own per-context handle namespace. Whenever a transient resource is created by a TPM command, TBS assigns a handle that uniquely identifies this resource within the caller's context. TBS replaces the handle in the original TPM response, and passes the modified response back to the application. Similarly, when an application sends a TPM command to TBS, TBS will replace the handles with the currently valid TPM handles for the referenced resources (after potentially context-loading these resources). This resource management, combined with the resource handle replacement, is referred to as *resource virtualization*.

The following table lists the type of resources that the TBS manages and the corresponding actions:

Resource Type	Handle Type	TBS Action
TPM1.2 Resources		Resource virtualization
TPM2.0 HMAC Session	0x02	Resource management
TPM2.0 Policy Session	0x03	Resource management
TPM2.0 Transient Object	0x80	Resource virtualization

Command Filtering for 1.2 and 2.0

When inspecting the TPM commands sent by applications, TBS will also apply filtering based on the TPM command code of the command. The TBS will block a command from running according to the following criteria:

- TPM 1.2: If the command code is on one of the blocked lists, the command will be blocked. Note that these lists can be modified using the appropriate WMI interfaces or using the Windows TPM management console.
- TPM 2.0: If the command code is not on one of the allow lists, and the command code does not specify a vendor-specific command, the command will be blocked. Vendor-specific commands will be allowed.

Note that the above blocked and allow lists differentiate between standard users and users with administrative access, and can be modified by Group Policy. In the standard Windows configuration, all TPM commands required for attestation are allowed.

TBS API

TBS provides the following function calls that are relevant for attestation. The functions are described in detail in the following sections.

```
Tbsi_Context_Create
Tbsip_Context_Close
Tbsi_Get_TCG_Log
Tbsi_Revoke_Attestation
Tbsi_GetDeviceInfo
Tbsip_Submit_Command
```

Creating TPM 1.2 and 2.0 Contexts

Most TBS functions require a TBS context, an object used by Windows to manage virtualized TPM resources. The TBS context serves as a handle to these TBS functions.

An application creates a TBS context using the **Tbsi_Context_Create()** function:

```
TBS_RESULT WINAPI Tbsi_Context_Create(
    __in  PCTBS_CONTEXT_PARAMS pContextParams,
    __out PTBS_HCONTEXT        phContext);
```

Parameters:

pContextParams [in] – A parameter to a **TBS_CONTEXT_PARAMS** structure that contains the parameters associated with the context. See the Remarks section below.

phContext [out] – A pointer to a location to store the new context handle.

Return value:

If the function succeeds, the function returns TBS_SUCCESS (0).

Remarks:

The *pContextParams* parameter allows the caller to specify the TPM version (TPM 1.2 or TPM 2.0) that it is prepared to interact with. For applications interacting with TPM version 1.2 only, a pointer to a **TBS_CONTEXT_PARAMS** can be provided, with the *version* field set to **TPM_VERSION_12**.

Applications interacting with TPM version 2.0 will pass a pointer to a **TBS_CONTEXT_PARAMS2** structure, with the *version* field set to **TPM_VERSION_20**. Set the *reserved* field to 0, and the *includeTPm20* field to 1. If the application is prepared to interact with TPM version 1.2 as well (in case the system has no TPM version 2.0), set the *includeTpm12* field to 1.

```
#define TPM_VERSION_12          1
#define TPM_VERSION_20          2

typedef struct {
    UINT32 version;
} TBS_CONTEXT_PARAMS, *PTBS_CONTEXT_PARAMS;
typedef const TBS_CONTEXT_PARAMS *PCTBS_CONTEXT_PARAMS;

typedef struct {
    UINT32  version;
    UINT32  reserved : 1;
    UINT32  includeTpm12 : 1;
    UINT32  includeTpm20 : 1;
} TBS_CONTEXT_PARAMS2, *PTBS_CONTEXT_PARAMS2;
typedef const TBS_CONTEXT_PARAMS2 *PCTBS_CONTEXT_PARAMS2;
```

If no TPM is present on the system, or the TPM version does not match those requested by the caller, **Tbsi_Context_Create()** will return the **TBS_E_TPM_NOT_FOUND** (0x8028400f) error code. Application programs must check the TPM version and be able to interact with either.

Deleting TPM 1.2 and 2.0 Contexts

An application can free a TBS context and release the associated system resources using the **Tbsip_Context_Close()** function:

```
TBS_RESULT WINAPI Tbsip_Context_Close(
    __in TBS_HCONTEXT   hContext);
```

Parameters:

hContext [in] – TBS context handle to free.

Return value:

If the function succeeds, the function returns TBS_SUCCESS (0).

Obtaining the Windows Boot Configuration Log (WBCL)

An application can obtain the WBCL using the Tbsi_Get_TCG_Log() function:

```
TBS_RESULT WINAPI Tbsi_Get_TCG_Log(
    __in    TBS_HCONTEXT hContext,
    __out_bcount_part_opt(*pcbOutput, *pcbOutput)
            PBYTE        pabOutput,
    __inout PUINT32      pcbOutput);
```

Parameters:

hContext [in] – TBS handle obtained from a previous call to **Tbsi_Context_Create()**.

pabOutput [out] – A pointer to a location to store the WBCL. This parameter may be NULL to estimate the required buffer when the location pointed to by pcbOutput is also 0 on input.

pcbOutput [in,out] – A pointer to a location that, on input, specifies the size, in bytes, of the output buffer. If the function succeeds, this parameter, on output, receives the size, in bytes, of the data pointed to by pabOutput. Calling the **Tbsi_Get_TCG_Log()** function with a zero length buffer will return the size of the buffer required.

Return value:

If the function succeeds, the function returns TBS_SUCCESS (0). If the *Size* parameter is too small, the function returns TBS_E_INSUFFICIENT_BUFFER (0x80284005).

Invalidating the System Trust State

If the ELAM driver detects a policy violation (a rootkit, for example), it can invalidate the PCRs that indicated that the system was in a good state using the **Tbsi_Revoke_Attestation()** function:

```
TBS_RESULT WINAPI
Tbsi_Revoke_Attestation();
```

Parameters:

None.

Return value:

If the function succeeds, the function returns TBS_SUCCESS (0).

Remarks:

This function, executable by users with administrative rights, extends PCR[12] by an unspecified value and increments the event counter in the TPM. Both actions are necessary, so the trust is broken in all quotes that are created from here on forward. Because the PCRs are reset on hibernation and the extend to PCR[12] then will disappear, a gap in the event counter will indicate a broken chain of logs.

As a result, the WBCL files will not reflect the current state of the TPM for the remainder of the time that the TPM is powered up and remote systems will not be able to form trust in the security state of the system. Note that anti-malware systems will probably perform additional remediation or alerts, but the invalidation step is crucial if attestation is supported.

When the machine goes to hibernation and subsequently resumes, the above PCR extend will be lost, and the broken trust will not be reflected in the PCR measurements anymore. To address this, **Tbsi_Revoke_Attestation()** also increments the monotonic Event Counter located in the TPM. Further TPM attestation validations will notice a gap in the archived WBCL logs' boot counter values. Upon discovery of such a gap, attestation validation code should fail the validation, just as it would if other required events were not present in the log. Note that the counter in the TPM cannot be rolled back, and hence the missing WBCL cannot be constructed after the fact. The log is described in more detail in the section "Windows Boot Configuration Log."

Obtaining the TPM Version

An application can obtain the version of the TPM on the system using the **Tbsi_GetDeviceInfo()** function:

```
TBS_RESULT WINAPI Tbsi_GetDeviceInfo(
    __in               UINT32           Size,
    __out_bcount(Size) PVOID            Info);
```

Parameters:

Size [in] – Size of the Info memory location.

Info [out] – A pointer to a location to store the version information about the TPM. The location must be large enough to hold four 32-bit values. For details, see the Remarks section below.

Return value:

If the function succeeds, the function returns TBS_SUCCESS (0). If no TPM is present on the system, the function returns TBS_E_TPM_NOT_FOUND (0x8028400f). If the *Size* parameter is too small, the function returns TBS_E_BAD_PARAMETER (0x80284002).

Remarks:

Upon success, the location pointed to by Info will be set according to the following table:

Offset	Size	Comment
0	4	Version of the Info structure. Will be set to 1.
4	4	TPM version. Will be set to TPM_VERSION_12 or TPM_VERSION_20.
8	4	Reserved.
12	4	Reserved.

Submitting a Custom TPM Command

An application can submit a preformatted command to TBS to be run on the TPM using the **Tbsip_Submit_Command()** function:

```
TBS_RESULT WINAPI
Tbsip_Submit_Command(
    __in                    TBS_HCONTEXT         hContext,
    __in                    TBS_COMMAND_LOCALITY Locality,
    __in                    TBS_COMMAND_PRIORITY Priority,
    __in_bcount(cbCommand)  PCBYTE               pabCommand,
    __in                    UINT32               cbCommand,
    __out_bcount(*pcbResult) PBYTE               pabResult,
    __inout                 PUINT32              pcbResult);
```

Parameters:

hContext [in] – TBS handle obtained from a previous call to **Tbsi_Context_Create()**.

Locality [in] – Must be TBS_COMMAND_LOCALITY_ZERO (0).

Priority [in] – The priority level that the command should have.

pabCommand [in] – A pointer to a buffer that contains the TPM command to process.

cbCommand [in] – The length, in bytes, of the command.

pabResult [out] – A pointer to a buffer to receive the result of the TPM command.

pResultBufLen [in, out] – A pointer to an integer that, on input, specifies the size, in bytes, of the result buffer. On successful return, this value is set to the actual size of the TPM response, in bytes.

Return value:

If the function succeeds, the function returns TBS_SUCCESS (0). Other values indicate errors detected by the TBS during the running of the command. This function can succeed (indicating that the command was sent to the TPM and the TPM responded), but the TPM could have returned an error. In this case, this function returns TBS_SUCCESS, and the TPM failure code is returned as a standard TPM response code in the result buffer.

Windows Boot Configuration Log

Events recorded in TPM PCRs are available through TBS with the call Tbsi_Get_TCG_Log(). Prior to Windows 8 the log covered all components from system startup to BootMgr. In Windows 8, measurements have been extended all the way to the kernel launch. These measurements are optional and may be enabled with '`bcdedit -set {globalsettings} integrityservices enable`'. In addition to the integrity measurements, a TPM monotonic counter is used to put logs into temporal order. Optionally an AIK may be registered that provides integrity protection to the log.

By default the last 100 system boot logs and all associated resume logs are archived in the folder %SystemRoot% \logs\measuredboot. The number of retained logs may be set with the registry REG_DWORD value `PlatformLogRetention` under the key `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TPM`. A value of 0 will turn off log archival and a value of 0xffffffff will keep all logs.

Windows Integrity Measurements

An XML-encoded sample of a Windows Boot Configuration Log is provided for reference purposes.

SampleLog.xml

The Windows-defined events are a tuple of {Type, Length, Value}. In order to minimize the number of extend calls to the TPM, the event data is aggregated until a trust boundary is crossed. Aggregated events are extended in a container event of the type SIPAEVENT_TRUSTBOUNDARY, SIPAEVENT_LOADEDMODULE_AGGREGATION, SIPAEVENT_ELAM_AGGREGATION or SIPAEVENT_TRUSTPOINT_AGGREGATION. The event type is EVENT_TAG and the SHA-1 digest of the event payload is the extended digest. The Windows integrity measurements are extended to PCR[12,13,14] and event types are defined in WBCL.h.

OS Data Events in PCR[12]

This PCR will change on every boot and resume.

SIPAEVENT_BOOTCOUNTER	Only used for TPM 2.0: the power-up counter of the TPM.
SIPAEVENT_TRANSFER_CONTROL	Extended by BootMgr to indicate if control is transferred to WinLoad, WinResume or a boot application.
SIPAEVENT_APPLICATION_RETURN	A boot application has returned to BootMgr.
SIPAEVENT_BITLOCKER_UNLOCK	BitLocker protector type used for volume unlock.
SIPAEVENT_EVENTCOUNTER	Monotonic event counter in the TPM.

SIPAEVENT_COUNTERID	Only used for TPM 1.2: Counter ID – has to remain the same across hibernate and resume.
SIPAEVENT_BOOTDEBUGGING	Boot debugging enabled.
SIPAEVENT_OSKERNELDEBUG	OS kernel debugging enabled.
SIPAEVENT_CODEINTEGRITY	Code integrity enabled.
SIPAEVENT_TESTSIGNING	Test signing enabled.
SIPAEVENT_DATAEXECUTIONPREVENTION	NX enabled.
SIPAEVENT_SAFEMODE	Safe mode enabled.
SIPAEVENT_WINPE	Windows PE boot.
SIPAEVENT_OSDEVICE	OS device used for boot.
SIPAEVENT_SYSTEMROOT	System root on OS device.
SIPAEVENT_HYPERVISOR_LAUNCH_TYPE	HV launch properties.
SIPAEVENT_HYPERVISOR_PATH	HV launch properties.
SIPAEVENT_HYPERVISOR_IOMMU_POLICY	HV launch properties.
SIPAEVENT_DRIVER_LOAD_POLICY	Driver load policy.
SIPAEVENT_AUTHENTICODEHASH with SIPAEVENT_IMAGEBASE	Table that indicates at what base address a given binary was loaded.
SIPAEVENT_ELAM_KEYNAME with SIPAEVENT_ELAM_CONFIGURATION, SIPAEVENT_ELAM_POLICY, SIPAEVENT_ELAM_MEASURED	Name of the ELAM AV Vendor key and the SHA-1 digests of the data in the values Config, Policy and Measured within the key in the ELAM hive.

OS Module Events in PCR[13]

This PCR will remain static until there is a change to the set of boot-start kernel-mode driver binaries, such as add, remove or update.

SIPAEVENT_FILEPATH	Name and path of the binary loaded.
SIPAEVENT_IMAGESIZE	Size of the binary loaded.
SIPAEVENT_HASHALGORITHMID	Authenticode hash algorithm.
SIPAEVENT_AUTHENTICODEHASH	Authenticode digest.

SIPAEVENT_AUTHORITYISSUER	ASN.1 encoded issuer of the signing certificate.
SIPAEVENT_AUTHORITYSERIAL	Signing certificate serial number.
SIPAEVENT_AUTHORITYPUBLISHER	ASN.1 encoded publisher of the binary.
SIPAEVENT_AUTHORITYSHA1THUMBPRINT	SHA-1 thumbprint of the signing certificate.

OS Authority Events in PCR[14]

This PCR is used to record the public keys of authorities that sign OS components. We expect that it will seldom change because even if binaries get updated or signing certificates are renewed, as long as they are signed with the same key this measurement will not change. The data is sorted before it is extended to ensure that an out-of-order load will produce the same digest in the PCR.

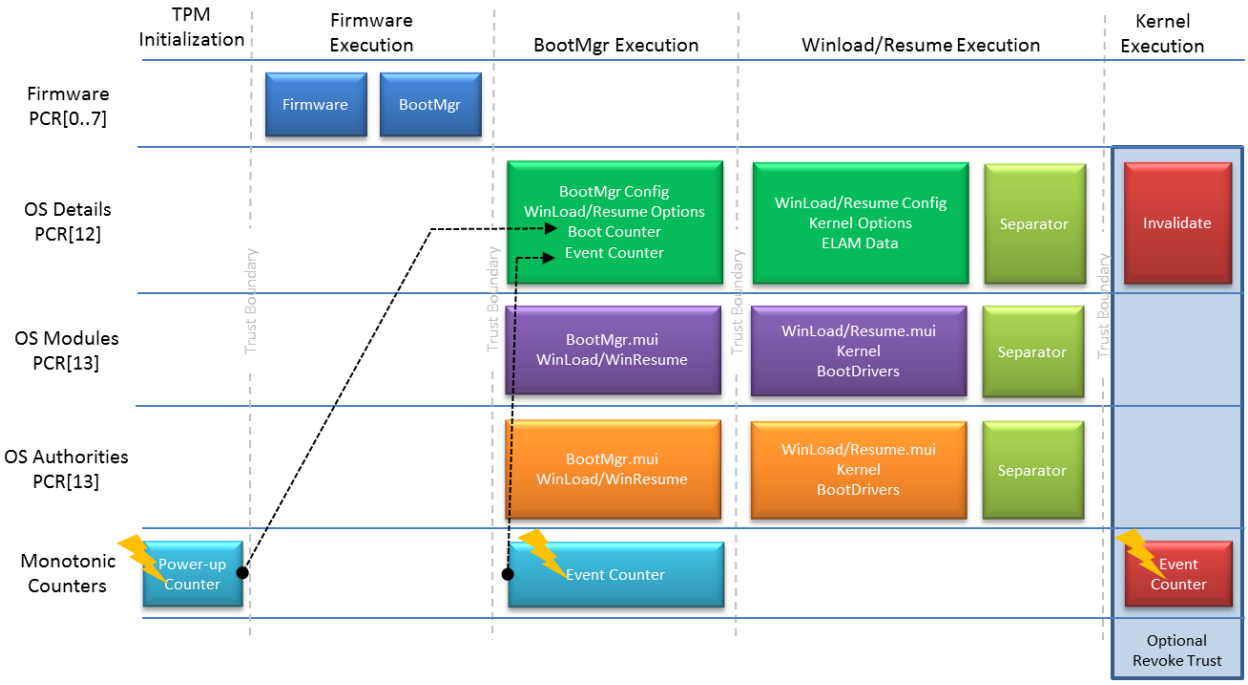
SIPAEVENT_AUTHORITYPUBKEY	ASN.1 encoded pubkey of certificate signer.
SIPAEVENT_NOAUTHORITY	Extended if an unsigned binary was run.

Non-Extended Events

An administrator can register one or more AIKs created with the Platform Crypto Provider and exported as BCRYPT_OPAQUE_KEY_BLOB in the registry under the key `HKLM\SYSTEM\CurrentControlSet\Services\Tpm\PlatformQuoteKeys`. The registry value name used to store the AIK should be the base64 encoded SHA1 thumbprint of the certificate for the AIK, to be able to locate the certificate for the key. On system start and hibernation resume when the TPM driver is initialized, all registered keys will be used to generate a quote over the log. The quote is appended to the log as a non-extended event and is archived with it. This will provide integrity protection to the log, while the event counter in the log will allow placing archived logs into temporal order for inspection at a later point in time.

SIPAEVENT_QUOTE	TPM_QUOTE_INFO2 Structure for TPM 1.2 and TPM2B_ATTEST for TPM 2.0.
SIPAEVENT_QUOTESIGNATURE	Signature over digest of SHA-1 digest of associated SIPAEVENT_QUOTE data.
SIPAEVENT_AIKPUBDIGEST	SHA-1 digest of the AIK modulus.
SIPAEVENT_AIKID	Value name of the AIK in the registry.

Root of Trust Overview



This graphic gives a brief overview of what PCR is extended and when the Monotonic counter is incremented by which executing instance on the system.

Platform Trust Considerations across Hibernation and Resume

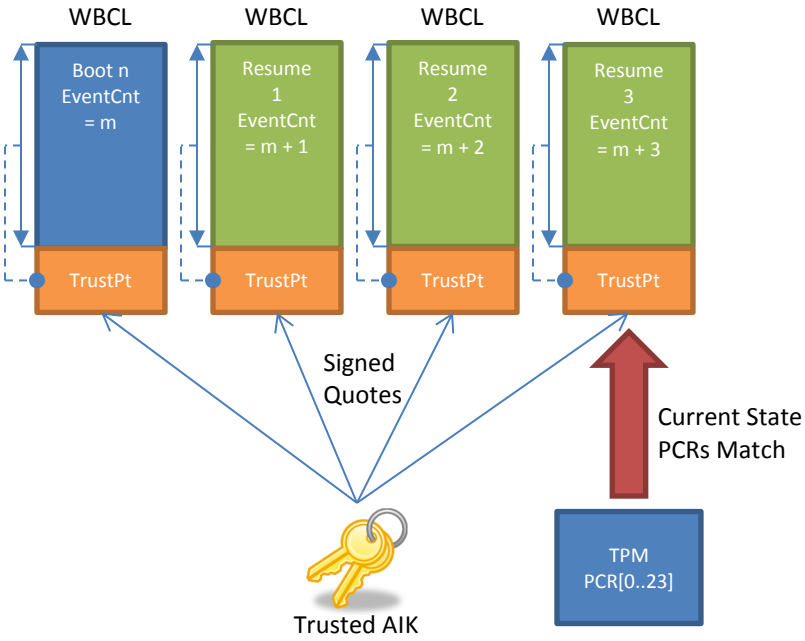
On a regular boot the platform firmware builds the WBCL and extends its measurements in PCR[0-7] and the OS components continue with boot binaries and drivers in PCR[8..10, 12..14]. A validator can inspect the log using an optional trust point in the log or have the client generate a platform attestation with a specified nonce, because the client remains in that state until the next reboot or hibernation.

Note that the state of the PCRs persists across sleep and therefore the WBCL remains valid.

When the machine goes into hibernation, the machine state is persisted in the swap file and the hibernation file. While on a BitLocker-protected system, these two files are protected against offline attacks from non-administrators, but a rogue administrator technically has the ability to alter the state of the system while it is in hibernation. If platform attestation is used to aid the administrator, and the administrator has no interest in compromising the confidential data or the security of the platform state, it can be assumed that trust across hibernation and resume is given.

On system resume, the platform boots up into BootMgr, as it does on any other regular boot. BootMgr will detect the hibernated OS and resume it by launching WinResume. WinResume will restore the memory image from the hibernation image and reanimate the OS. It is important to note that in that case

the boot drivers are *not* individually loaded and extended; hence a resume log will lack all the boot driver measurements that a full boot log provides.



In order to consider the trust of a resumed OS, it is important to validate not only the current WBCL of the system that is backed up by the PCRs in the TPM, but also the previous WBCLs that describe all resumes prior to the last one all the way to the last full boot that contains the boot driver details for this OS launch. Since the WBCLs of prior boots and resumes are archived on the OS volume, this inspection is possible. Potential malware that is aware of the WBCL log on the system may have access to the archived files and could attempt to alter them to hide its existence. Because the TPM has been rebooted, there is no direct way to prove that an archived WBCL has not been tampered with.

To protect the integrity of an archived WBCL, an optional trusted and registered AIK is used by the TPM driver to create a TPM quote when the system is initialized or re-initialized. A TPM quote is a signature over external data (in this case a hash of the boot log) together with PCR values at the time the quote was generated. Since an AIK is restricted and cannot sign arbitrary data, the signature over a quote that validates the WBCL may be validated even if the TPM has been rebooted. The event counter inside the log therefore acts as the nonce for the quote and puts the logs in order. The validator can now look at all the logs and track the event counter down to the last full boot WBCL, validating and inspecting all logs on the way.

Because malware cannot roll back the counter in the TPM or sign any arbitrarily constructed WBCL, tampering with an individual log or omitting a log will be immediately obvious.

ELAM Driver Data Measurements

The data consumed by the ELAM driver is measured if the ELAM driver consumes this data from certain locations in the new ELAM registry hive. This hive is mounted in Winload and present when the ELAM driver runs and unmounted when the ELAM driver is unloaded. The SHA-1 digests of three REG_BINARY registry values under every [AVVendor key] key are recorded in Winload:

a) ELAM\[*AVVendor key*]\Config
b) ELAM\[*AVVendor key*]\Policy
c) ELAM\[*AVVendor key*]\Measured

Automatic Key Certification for Platform-Bound Keys

Non-exportable keys are tremendously valuable for certificates that are supposed to be bound to the machine. However, normally a CA has no way to directly validate that a particular key in fact is non-exportable. With the functionality of the PCP KSP in Windows 8, it is now possible for a CA to determine this crucial property.

Similar to the registered AIKs that provide trust points and integrity protect the WBCL, AIKs may be registered to automatically generate key certification data for every non-exportable key that is generated on the provider. This certification data is persisted with the key and can be retrieved as a key property.

An administrator can register one or more AIKs created with the Platform Crypto Provider and exported as BCRYPT_OPAQUE_KEY_BLOB in the registry under the key `HKLM\SYSTEM\CurrentControlSet\Services\Tpm\KeyAttestationKeys`.

Format of the Key Attestation Data

The attestation data is a list of attestation structures preceded by a UINT32 value that indicates the type of attestation data. The first UINT32 in each structure gives the size of the individual attestation structure. Then the individual data follows with a USHORT size and a byte buffer. The data order is:

1. Zero terminated registry value name that the AIK is stored under.
2. AIK_{Pub} SHA-1 key digest.
3. Attestation blob and signature over the SHA-1 digest of the attestation blob.

There will be one entry in this list for every registered AIK. Below is an example with one AIK on a TPM 1.2 platform.

```
Attestation Tag:
000000e3`92fe4860 54 31 4b 41                                     T1KA
Attestation blob size:        86 01 00 00                             ....
AIK name:                                    16 00 44 00 65 00 66 00          ..D.e.f.
000000e3`92fe4870 61 00 75 00 6c 00 74 00 41 00 69 00 6b 00 00 00  a.u.l.t.A.i.k...
AIKPub Digest:
000000e3`92fe4880 14 00 1a 36 a1 e1 a7 6f 20 51 90 0e 50 c6 3b 36  ...6...o Q..P.;6
000000e3`92fe4890 ec e1 12 6e 47 dd
Attestation Data:                       50 00 01 01 00 00 00 12 00 00  ...nG.P.........
000000e3`92fe48a0 00 04 00 00 00 00 01 00 01 00 02 00 00 00 0c 00  ................
000000e3`92fe48b0 00 08 00 00 00 00 02 00 00 00 00 a9 10 71 10 3c  .............q.<
000000e3`92fe48c0 d7 f6 02 67 25 58 37 02 a6 ce a1 ae a3 85 2b 00  ...g%X7.......+.
000000e3`92fe48d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000000e3`92fe48e0 00 00 00 00 00 00 00 00
Attestation Signature:                              00 01 39 26 0e 2a 72 3a  ..........9&.*r:
000000e3`92fe48f0 64 45 d6 06 71 f6 ef 21 60 56 71 78 45 b3 3d fd  dE..q..!`VqxE.=.
000000e3`92fe4900 7e 4f 67 bf 24 f5 54 b8 44 20 5c c4 d9 d9 1e 2b  ~Og.$.T.D \....+
000000e3`92fe4910 83 01 3d 8a 5b 14 eb 0f 2a 46 f4 e0 b2 87 16 e6  ..=.[...*F......
000000e3`92fe4920 9f e6 54 f0 02 2a 7e 7e a9 ec d8 a0 e9 b9 54 6d  ..T..*~~......Tm
000000e3`92fe4930 01 39 9e a3 f3 7b a4 47 d6 be 64 b7 56 53 e5 51  .9...{.G..d.VS.Q
000000e3`92fe4940 7b 5e 26 f8 34 2e bd ef 77 69 cc 00 10 93 ed 16  {^&.4...wi......
000000e3`92fe4950 1f a1 8d c3 2c 1a 68 bb e6 fd e7 cd 74 15 5d aa  ....,.h.....t.].
000000e3`92fe4960 ed 87 91 41 79 fd fe d3 86 d5 00 4c f6 22 22 22  ...Ay......L."""
000000e3`92fe4970 dd 60 ed 46 11 93 1a b2 6a c4 1a 49 79 1d 4c e9  .`.F....j..Iy.L.
000000e3`92fe4980 67 db 7b c3 8f 6a 5d 56 22 6c 14 20 43 3d 06 26  g.{..j]V"l. C=.&
000000e3`92fe4990 f0 57 ec 68 16 5d d9 4b ba 55 d8 9f 94 de 5f 34  .W.h.].K.U...._4
000000e3`92fe49a0 6e 29 e4 90 46 56 94 df 04 20 38 ee 8e 35 c6 09  n)..FV... 8..5..
000000e3`92fe49b0 09 92 44 57 df 20 60 27 f9 b6 b3 23 57 fc 0d 26  ..DW. `'...#W..&
000000e3`92fe49c0 52 5a 91 08 2d 24 0e e1 65 55 bb 08 bc 72 81 cd  RZ..-$..eU...r..
000000e3`92fe49d0 30 3b 74 09 be 3e 48 a3 bf 51 bf b6 b6 30 2d 26  0;t..>H..Q...0-&
000000e3`92fe49e0 b7 dd ce 97 2a 56 3f 07 07 17
```

Attestation API Reference Implementation

Attestation is a very complex area that requires significant detailed knowledge about the platform or entity that seeks attestation. Typically, attestation scenarios demand physical or logical separation between client and validator to ensure trustworthy evaluation of the attestation data. For that reason this document provides the source code of a library that provides components to perform client-side attestation and server-side validation for TPM 1.2 and 2.0.

Introduction

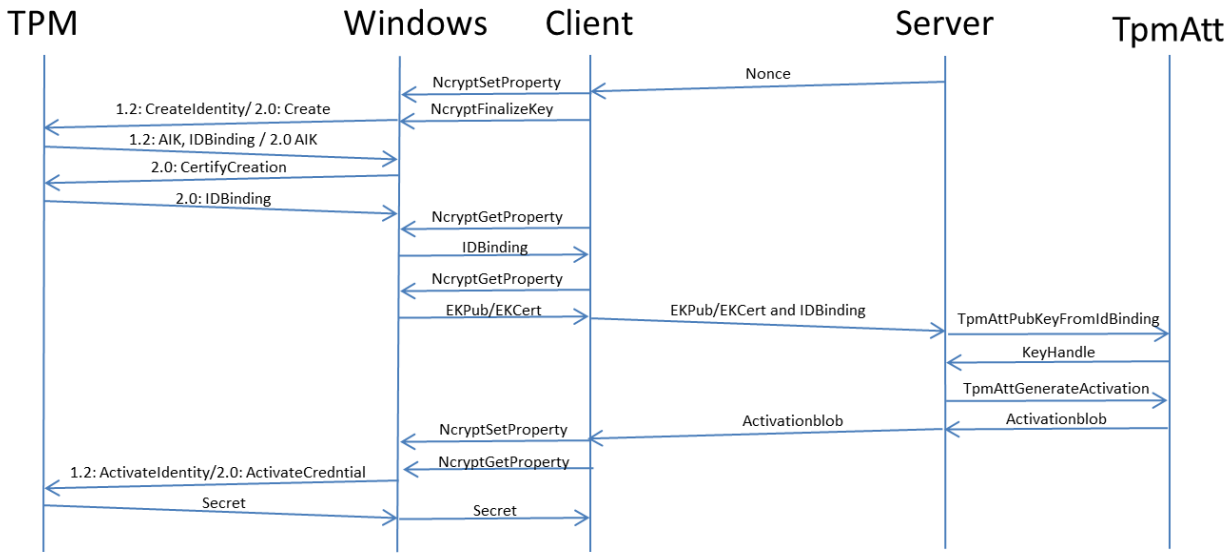
The attestation library works closely with the Platform Crypto Provider, the TPM driver and TPM Base Services (TBS) and implements all required functionality that is not directly provided by Windows 8. Only publicly available Windows 8 interfaces are used. The server-side API has very little dependence on the BCrypt RSA and AES crypto providers and memory allocation.

The library is provided with this document as a DLL, but vendors may choose other distribution mechanisms.

This section will introduce the individual API calls that are provided, grouped by scenario.

Creating Attestation Identity Keys (AIKs) and Forming Remote Trust

The process is started by the server, which sends a key creation nonce to the client. The client opens the Platform Crypto Provider and starts the key creation process. The client sets the new key's property `NCRYPT_PCP_KEY_USAGE_POLICY` to `NCRYPT_PCP_IDENTITY_KEY` and provides the 20-byte nonce from the server as property `NCRYPT_PCP_TPM12_IDBINDING`. The client then finalizes the key, which causes the TPM to create an AIK on TPM 1.2 and a restricted signing key on TPM 2.0. On TPM 1.2, this command also creates the IDBinding, while on TPM 2.0, certify creation is called to generate the equivalent data structure. The IDBinding is retrieved from the Platform Crypto Provider by reading the property `NCRYPT_PCP_TPM12_IDBINDING` and the result is sent to the server for validation and to create the challenge with the EK for the TPM.



TpmAttPubKeyFromIdBinding()

This function is used on the server to extract the AIK_{PUB} from the ID binding and get a BCrypt handle to it. The AIK_{PUB} handle may be handed to an enrollment agent to export the public key and create the AIK Certificate. It is important to note that, since this is just a handle to the AIK_{PUB}, this key will not be able to sign a certificate request to show proof of possession.

This function does not use a TPM.

```
HRESULT
TpmAttPubKeyFromIdBinding(
    _In_reads_(cbIdBinding) PBYTE pbIdBinding,
    UINT32 cbIdBinding,
    BCRYPT_ALG_HANDLE hRsaAlg,
    _Out_ BCRYPT_KEY_HANDLE* phAikPub
    );
```

Parameters:

`IdBinding` – Buffer and size containing the opaque IDBinding data that the client has retrieved from the property `NCRYPT_PCP_TPM12_IDBINDING` after the key was finalized. The API can identify IDBindings from TPM 1.2 and 2.0 and create the corresponding activation credential blob. The IDBinding accepted by this function consists of a structure containing the public portion of the AIK along with other data, and a signature over this structure.

`hRsaAlg` – Handle to the RSA BCrypt provider that should be used to import the public key.

`phAikPub` – Pointer that will receive the BCrypt key handle of the AIK_{PUB}.

TpmAttGenerateActivation()

This function is used on the server to generate the encrypted challenge for the TPM to show that the EK and AIK reside in the same TPM and are valid. The server has the responsibility to validate the EK_{PUB}

and establish trust prior to executing this operation (very likely by inspecting the EK certificate and its chain of trust or looking up the EK_{PUB} on a list of trusted or well-known EKs). Handling the EK certificate is not part of this API and can be done with Windows PKI APIs.

```
HRESULT
TpmAttGenerateActivation(
    BCRYPT_KEY_HANDLE hEkPub,
    _In_reads_(cbIdBinding) PBYTE pbIdBinding,
    UINT32 cbIdBinding,
    _In_reads_opt_(cbNonce) PBYTE pbNonce,
    UINT32 cbNonce,
    _In_reads_(cbSecret) PBYTE pbSecret,
    UINT16 cbSecret,
    _Out_writes_to_opt_(cbOutput, *pcbResult) PBYTE pbOutput,
    UINT32 cbOutput,
    _Out_ PUINT32 pcbResult
    );
```

If the IDBinding was produced by a TPM 1.2, the function will place the provided secret with the digest of the AIK_{PUB} in a TPM_EK_BLOB_ACTIVATE structure contained in a TPM_EK_BLOB structure that is OAEP-encrypted with the EK_{PUB}. No TPM is required for this operation.

If the AIK was created on a TPM 2.0, the equivalent functionality to the TPM2_CreateCredential is implemented in software. Again, no TPM on the server side is required for this operation.

Parameters:

`hEkPub` – BCrypt handle to the EK_{PUB} that will be used to encrypt the credential.

`IdBinding` – Data blob produced when the key is finalized in the Platform Crypto Provider. This blob is TPM version-specific.

`Nonce` – Optional 20-byte value provided by the server to ensure that a new AIK was created.

`Secret` – 20-byte secret that the server wants to use as a credential. This could be a token or a symmetric key that encrypts a larger secret.

`Output, Result` – Encrypted credential or activation blob.

Obtaining and Parsing Platform Configuration and Measurements

TpmAttGetPlatformCounters()

```
HRESULT
TpmAttGetPlatformCounters(
    _Out_opt_ PUINT32 pOsBootCount,
    _Out_opt_ PUINT32 pOsResumeCount,
    _Out_opt_ PUINT64 pCurrentTpmBootCount,
    _Out_opt_ PUINT64 pCurrentTpmEventCount,
    _Out_opt_ PUINT64 pCurrentTpmCounterId,
    _Out_opt_ PUINT64 pInitialTpmBootCount,
    _Out_opt_ PUINT64 pInitialTpmEventCount,
    _Out_opt_ PUINT64 pInitialTpmCounterId
    );
```

This function is used on the client to obtain the current counter values from the platform.

Parameters:

`OsBootCount` – Pointer to the location where the OS boot counter is to be stored. This counter is not protected and is used as an index to the log archive. This counter is incremented on every OS boot.

`OsResumeCount` – Pointer to the location where the OS resume counter is to be stored. This counter is not protected and is used as an index to the log archive. This counter is reset on every fresh boot and incremented on every resume from hibernate.

`CurrentTpmBootCount` – Pointer to the location where the TPM boot counter is to be stored. This counter reflects the TPM 2.0 power-up counter. This is a monotonic counter and is protected in the TPM. On TPM 1.2 this counter is 0. A validator may validate that this counter is contiguous to ensure that a platform was not powered up and booted from different media in between hibernation and resume to tamper with the hibernation data. This counter does not detect if a physically present administrator mounted the OS volume in a different machine.

`CurrentTpmEventCount` – Pointer to the location where the TPM event counter is to be stored. This counter is incremented every time BootMgr is run and reflects the monotonic event counter in the TPM. A validator can use this counter to verify that a set of boot logs is contiguous from a cold boot log including all resume logs.

`CurrentTpmCounterId` – Pointer to the location where the TPM counter ID of the TPM 1.2 is to be stored. On TPM 2.0 this ID never changes. If this value changes, the validator knows that a different monotonic counter in the TPM was used and indicates a possible attack.

`Initial*` – These values are recorded the last time the platform was cold-booted. The validator may use these values as references when validating a set of logs. These values are stored in the registry and are not protected.

TpmAttGetPlatformLogFromArchive()

```
HRESULT
TpmAttGetPlatformLogFromArchive(
    UINT32 OsBootCount,
    UINT32 OsResumeCount,
    _Out_writes_to_opt_(cbOutput, *pcbResult) PBYTE pbOutput,
    UINT32 cbOutput,
    _Out_ PUINT32 pcbResult
    );
```

This function may be used to look up a specific log from the archive and return its content. The archive is not integrity-protected and all logs received from there need to be validated before the content is used to derive a trust decision. It is possible to go back in time beyond the last full boot and inspect previous boots. By default, the machine keeps the last 100 cold boots with all associated resume logs in the archive. It is possible to alter this behavior by setting with the registry REG_DWORD value `PlatformLogRetention` under the key `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TPM` with the number of full boots to retain.

Parameters:

`OsBootCount` – OS boot count. This counter is unprotected and is not to be confused with any of the TPM counters.

`OsResumeCount` – OS Resume count. This counter is unprotected and is not to be confused with any of the TPM counters.

`Output, Result` – Retrieved log data.

Platform Attestation and Validation

TpmAttGeneratePlatformAttestation()

```
HRESULT
TpmAttGeneratePlatformAttestation(
    NCRYPT_KEY_HANDLE hAik,
    UINT32 pcrMask,
    _In_reads_opt_ (cbNonce) PBYTE pbNonce,
    UINT32 cbNonce,
    _Out_writes_to_opt_(cbOutput, *pcbResult) PBYTE pbOutput,
    UINT32 cbOutput,
    _Out_ PUINT32 pcbResult
    );
```

This function is used to generate a platform attestation on the client system, and is typically called using a nonce from the server (to guarantee freshness and to protect against attestation state replay). This function uses the TPM on the client. It requires an NCrypt handle to the AIK that is used to sign the quote command. The quote will contain PCR values for all PCRs specified by the caller. The difference from

the quote in the trust point is that the server may specify a nonce here that is used in the quote and will act as proof that the client was in this state at the time the quote was signed. (There is no guarantee that the client remained in this state at the time the server is evaluating the platform attestation.)

If the AIK requires authorization, it has to be fully authorized before the user calls this function.

Parameters:

`hAik` – Optional NCrypt handle to the fully authorized AIK in the Platform Crypto Provider. If no handle is provided, the attestation blob will not contain a quote, just the PCRs that are read from the TPM. (In this case the attestation blob is not integrity-protected.)

`pcrMask` – Mask of PCRs to be included in the quote. Bit 0 for the 32-bit value corresponds with PCR[0] up to bit 23 with PCR[23].

`Nonce` – Optional 20-byte value to be included in the quote, so the server can validate that the quote was generated for a particular server request.

`Output, Result` – AIK Signed Quote structure followed immediately by the signature data. The signature size of the AIK defines the size of the signature following the TPM_QUOTE_INFO or TPM_QUOTE_INFO2 on TPM 1.2 or TPM2B_ATTEST structure on TPM 2.0. The signature may be validated without being able to understand the TPM-generated structure by calculating the SHA-1 digest of pbOutput[0..*pcbResult – 1 – sizeof(AIK Signature size)] and making an RSA signature validation with the calculated digest and the signature in pbOutput[*pcbResult – sizeof(AIK Signature size)..*pcbResult – 1].

TpmAttCreateAttestationfromLog()

```
HRESULT
TpmAttCreateAttestationfromLog(
    _In_reads_(cbLog) PBYTE pbLog,
    UINT32 cbLog,
    _In_reads_z_(MAX_PATH) PWSTR szAikNameRequested,
    _Outptr_result_z_ PWSTR* pszAikName,
    _Out_writes_to_opt_(cbOutput, *pcbResult) PBYTE pbOutput,
    UINT32 cbOutput,
    _Out_ PUINT32 pcbResult
    );
```

This function may be used on the client or the server and does not use the TPM. It is used to turn an archived log file into an attestation structure that may be validated like a quote that was generated with the call `TpmAttGeneratePlatformAttestation()`. This function will succeed only if an AIK was registered and the trust point was successfully written to the log.

The difference here is that the nonce in trust point was not generated by a server and therefore only proves that the client at some point was in the specified configuration. This function may only be used to verify the chain from the current log across all resume logs back to the last cold boot log. Once all these archived logs have been validated, the server can issue a nonce and ask the client to perform a platform attestation with the nonce and simply compare that the PCRs in that attestation match the PCRs in the last

archived log, which indicates that the events in PCR[12] are identical and therefore the event counter has not been incremented.

Parameters:

`Log` – Buffer that contains the WBCL with the trust point in it.

`AikNameRequested` – Optional parameter that may be used to specify the name of the registry value that contains the AIK. If no name is specified, the first trust point is used.

`AikName` – AIK name of the trust point used for the quote. This may be used to locate an AIK Certificate on the system, for example.

`Output, Result` – Attestation structure.

TpmAttValidatePlatformAttestation()

```
HRESULT
TpmAttValidatePlatformAttestation(
    BCRYPT_KEY_HANDLE hAik,
    _In_reads_opt_ (cbNonce) PBYTE pbNonce,
    UINT32 cbNonce,
    _In_reads_ (cbAttestation) PBYTE pbAttestation,
    UINT32 cbAttestation
    );
```

This command is used on the server to perform an integrity validation of an attestation structure. This command has to be run in a trusted environment. (If run on the local client, malware could tamper with the presented data or fake the running of this code and hide the fact that the machine is not in a trusted state.)

The call will validate the signature with the provided AIK_{PUB} and verify the nonce in the quote (if one was provided). Further, this function will calculate the PCRs expected from the log contained in the attestation structure and check that they match the PCRs in the quote.

If this function succeeds, the caller may trust the WBCL log entries as much as the caller trusts the AIK.

But there is one caveat: The EK certificate that may have been used to establish trust in the AIK only speaks about the TPM and not about the system the TPM is bound to. So a rogue may use a TPM on wires connected to a USB port that is fully under that person's control and sign good quotes with an AIK that was only trusted because of its EK certificate. There is a missing link from the EK and the EK cert in the TPM to the actual platform itself. So far OEMs have not provided EK platform certificates that show that a given EK resides in a TPM that is actually bound to a system in the TCG-prescribed way.

In enterprise scenarios, the missing link may be provided by a system administrator who will issue enterprise EK certificates to all assets an enterprise recognizes as trustworthy. In an unmanaged environment, the remote party can never form full trust in the physical machine and therefore attestation always requires that the user or administrator of the machine have a vested interest to not fake measurements

Parameters:

`hAik` – BCrypt handle to the trusted AIK that will be used to validate the signature of the quote.

`Nonce` – Optional nonce sent from the server to ensure that the quote was produced for a particular request.

`Attestation` – Attestation data structure from the client.

TpmAttGetPlatformAttestationProperties()

```
HRESULT
TpmAttGetPlatformAttestationProperties(
    _In_reads_(cbAttestation) PBYTE pbAttestation,
    UINT32 cbAttestation,
    _Out_opt_ PUINT64 pEventCount,
    _Out_opt_ PUINT64 pEventIncrements,
    _Out_opt_ PUINT64 pEventCounterId,
    _Out_opt_ PUINT64 pBootCount,
    _Out_opt_ PUINT32 pdwPropertyFlags
    );
```

This call is used on the server to get properties from the attestation after it was validated.

Parameters:

`Attestation` – Attestation data structure.

`EventCount` – Pointer to the location that will receive the initial event count number of the log.

`EventIncrements` – Event increments in the log. By default, only one.

`EventCounterId` – Event counter ID in the log.

`BootCount` – TPM 2.0 power-up counter from the log.

`PropertyFlags` – Property flags from the log:

```
#define PCP_ATTESTATION_PROPERTIES_CONTAINS_BOOT_COUNT (0x00000001)
#define PCP_ATTESTATION_PROPERTIES_CONTAINS_EVENT_COUNT (0x00000002)
#define PCP_ATTESTATION_PROPERTIES_EVENT_COUNT_NON_CONTIGUOUS (0x00000004)
#define PCP_ATTESTATION_PROPERTIES_INTEGRITY_SERVICES_DISABLED (0x00000008)
#define PCP_ATTESTATION_PROPERTIES_TRANSITION_TO_WINLOAD (0x00000010)
#define PCP_ATTESTATION_PROPERTIES_TRANSITION_TO_WINRESUME (0x00000020)
#define PCP_ATTESTATION_PROPERTIES_TRANSITION_TO_OTHER (0x00000040)
#define PCP_ATTESTATION_PROPERTIES_BOOT_DEBUG_ON (0x00000100)
#define PCP_ATTESTATION_PROPERTIES_OS_DEBUG_ON (0x00000200)
#define PCP_ATTESTATION_PROPERTIES_CODEINTEGRITY_OFF (0x00000400)
#define PCP_ATTESTATION_PROPERTIES_TESTSIGNING_ON (0x00000800)
#define PCP_ATTESTATION_PROPERTIES_BITLOCKER_UNLOCK (0x00001000)
#define PCP_ATTESTATION_PROPERTIES_OS_SAFEMODE (0x00002000)
#define PCP_ATTESTATION_PROPERTIES_OS_WINPE (0x00004000)
#define PCP_ATTESTATION_PROPERTIES_OS_HV (0x00008000)
```

It is important to note that if PCP_ATTESTATION_PROPERTIES_INTEGRITY_SERVICES_DISABLED is set, the log does not contain any information about the defined properties below it.

Key Attestation and Validation

Key attestation (called key certification in TCG specifications) allows a user to prove to a remote entity certain properties of a TPM-protected key so that relying parties can be assured that it meets requirements for its intended use. One important use of this capability is securely and remotely provisioning certificates for signing keys.

For example, consider remote provisioning of a TPM key for use as a virtual smart card. Without the capabilities described in this section, a remote entity cannot determine whether a received request for a certificate is for a key that is actually TPM-protected, and not a rootkit or other malware that has created a software key that it could export or otherwise misuse.

Key attestation uses an AIK to sign properties of another TPM key, including whether the key is non-migratable and was created on this TPM. Hence, if an enterprise or other authority has securely provisioned the AIK to known good assets, this allows them to establish that the key is indeed TPM-protected and trustworthy.

Key attestation is strong, even without a platform certificate, because the attestation is only speaking about assets that remain inside the TPM. It does not matter how the TPM is bound to the system. A non-exportable key is confined to the TPM and this guarantee is not affected by a TPM that is in the physical control of a user. If key attestation is used in a certificate enrollment, the administrator will get strong proof from the TPM manufacturer that a given key adheres to the policy that the administrator requires. This proof effectively eliminates the requirement for the administrator to physically hold the device to be sure that the key is correctly provisioned and allows the user to remotely self-serve such a certificate directly with the CA.

TpmAttGenerateKeyAttestation()

This function uses a TPM 1.2 or TPM 2.0 to create an AIK-signed attestation structure from a second TPM-held key. The server may seed this attestation with a nonce to ensure that the attestation was generated upon the server's request.

```
HRESULT
TpmAttGenerateKeyAttestation(
    NCRYPT_KEY_HANDLE hAik,
    NCRYPT_KEY_HANDLE hKey,
    _In_reads_opt_(cbNonce) PBYTE pbNonce,
    UINT32 cbNonce,
    _Out_writes_to_opt_(cbOutput, *pcbResult) PBYTE pbOutput,
    UINT32 cbOutput,
    _Out_ PUINT32 pcbResult
    );
```

Both keys have to be loaded under the same Platform Crypto Provider handle and the keys have to be properly authorized for private key use (if they are using PINs). The function will return an attestation structure that usually contains a TPM_CERTIFY_INFO2 for TPM 1.2 (unless the key authorization

requirements of the keys force the API to create a TPM_ CERTIFY_INFO structure). On TPM 2.0, it will contain a TPM2B_ATTEST structure.

Parameters:

`hAik` – Fully authorized NCrypt handle that will be used to sign the key certification in the TPM.

`hKey` – Fully authorized NCrypt handle from the same provider as hAIK of the key that is to be attested.

`Nonce` – Optional server-provided nonce to ensure that the attestation was created upon the server's request.

`Output, Result` – Attestation data structure.

TpmAttCreateAttestationfromKey()

This function may be used to retrieve a key attestation structure for a key that was attested by the crypto provider at creation time for the purpose of validating that attestation with the function `TpmAttValidateKeyAttestation()`.

The preconditions for this function to succeed are that a valid AIK was registered at the time the key was finalized, and that the key was created inside the TPM and is non-exportable.

```
HRESULT
TpmAttCreateAttestationfromKey(
    NCRYPT_KEY_HANDLE hKey,
    _In_reads_z_(MAX_PATH) PWSTR szAikNameRequested,
    _Out_writes_z_(MAX_PATH) PWSTR szAikName,
    _Out_writes_to_opt_(cbOutput, *pcbResult) PBYTE pbOutput,
    UINT32 cbOutput,
    _Out_ PUINT32 pcbResult
    );
```

Parameters:

`hKey` – Fully authorized NCrypt handle of the key for which the attestation information is supposed to be retrieved.

`AikNameRequested` – Optional parameter that may be used to specify the name of the registry value that contains the AIK. If no name is specified, the first trust point is used.

`AikName` – AIK name of the trust point used for the quote. This may be used to locate an AIK certificate on the system, for example.

`Output, Result` – Attestation data structure.

TpmAttValidateKeyAttestation()

This API may be used to validate that a key is locally generated, is non-exportable, and (optionally) the specific PCR configuration that the key is accessible in.

PCR binding of keys can be useful to implicitly prove trust in a configuration when using legacy protocols. For example, a server might issue a TPM-protected machine key to a specific trustworthy configuration (as represented by a validated set of PCR values). Since the key may only be used as long as the client remains in the state that was validated by the server, use of this key is proof that the system is in the same configuration as when provisioned.

This scenario requires that the third party have a trust relationship with the health evaluation server. The client has to generate a new key every time the machine resumes from hibernation, and the server has to issue a new health certificate at the same time.

```
HRESULT
TpmAttValidateKeyAttestation(
    BCRYPT_KEY_HANDLE hAik,
    _In_reads_opt_(cbNonce) PBYTE pbNonce,
    UINT32 cbNonce,
    _In_reads_(cbAttestation) PBYTE pbAttestation,
    UINT32 cbAttestation,
    UINT32 pcrMask,
    _In_reads_opt_(cbPcrTable) PBYTE pcrTable,
    UINT32 cbPcrTable
    );
```

This is the server-side API to validate a key certification. The API discovers automatically whether the attestation was created on a TPM 1.2 or 2.0 and then first, it performs the integrity validation of the attestation; second, it verifies that the public key included in the attestation structure matches the one that was attested; and third, it verifies that the key is bound to the optionally provided PCR policy.

Parameters:

`hAik` – BCrypt handle to the AIK that will be used to validate the integrity protection in the attestation structure.

`Nonce` – Optional nonce that was assigned by the server to ensure that the attestation was issued upon the server's request.

`Attestation` – Attestation data structure from the client machine.

`pcrMask` – PCR bitmask that the key is supposed to be bound to. PCR[0] corresponds to bit 0 up to bit 23 that corresponds to PCR[23].

`pcrTable` – Full set of all 24 SHA-1 PCRs that are used to calculate the PCR policy to which the key is supposed to adhere.

TpmAttGetKeyAttestationProperties()

This function is used on the server side to obtain specific key properties to be validated against the policy. This function is also used to obtain a BCrypt key handle to the attested key. The server may use that BCrypt handle to create a health certificate, for example.

```
HRESULT
TpmAttGetKeyAttestationProperties(
    _In_reads_(cbAttestation) PBYTE pbAttestation,
    UINT32 cbAttestation,
    _Out_opt_ PUINT32 pPropertyFlags,
    BCRYPT_ALG_HANDLE hAlg,
    _Out_opt_ BCRYPT_KEY_HANDLE* phKey
    );
```

The function will parse the attestation structure and set property flags. If the caller desires, it will also return a BCrypt key handle.

Parameters:

`Attestation` – Attestation data structure from the client.

`PropertyFlags` – Pointer to a 32-bit value that will receive the property flags:

```
#define PCP_KEY_PROPERTIES_NON_MIGRATABLE (0x80000000)
#define PCP_KEY_PROPERTIES_PIN_PROTECTED (0x40000000)
#define PCP_KEY_PROPERTIES_PCR_PROTECTED (0x20000000)
#define PCP_KEY_PROPERTIES_SIGNATURE_KEY (0x00000001)
#define PCP_KEY_PROPERTIES_ENCRYPTION_KEY (0x00000002)
#define PCP_KEY_PROPERTIES_GENERIC_KEY (0x00000003)
#define PCP_KEY_PROPERTIES_STORAGE_KEY (0x00000004)
#define PCP_KEY_PROPERTIES_IDENTITY_KEY (0x00000005)
```

`hAlg` – BCrypt algorithm provider that should be used to import the public key.

`hKey` – Pointer to the location that will receive the BCrypt handle to the public key.

Key Hostage

The key "hostage" scenario is a variation of the scenario described in the section "Key Attestation and Validation." The main difference between the trust model and the hostage model is that in the hostage model the server creates the key and binds it to a specific PCR configuration. The server-generated key can be used only by the intended TPM when the platform is in the intended configuration.

Key hostage may be used to distribute to a particular machine keys that can only be used if the receiving machine meets a particular configuration policy. The recipient TPMs can use the keys if the policy is met, but cannot forward them (unless specifically authorized).

Another example would be a PIN-protected key for which the user has lost the PIN. The user can ask the server to reissue the key with a new PIN (after authenticating to the server) for use on the local TPM on the machine.

In the health certificate scenario described above, the difference would be that the server would generate and hold on to a key pair for each client and issue a health certificate for it. The client now would send only the WBCL to the server for inspection. If the server likes the WBCL, the server would wrap the health key for the current PCRs derived from the WBCL. The client would import the key blob and could immediately start using it (as long as the PCRs are set as the WBCL claimed). This optimization

eliminates the need for all expensive TPM operations on the client side: no platform attestation, no client-side key generation, and no key attestation required.

However, at enrollment time the client has to provide the SRK_{PUB} to the server. This could be done securely with a single key attestation.

TpmAttWrapPlatformKey()

This function is intended to be used on the server to wrap a server-held key for a particular TPM. While the API supports creation of keys for TPM 1.2 and 2.0, it has no means to identify for which TPM version a key is supposed to be wrapped. The caller may maintain a database of trusted SRK_{PUB} with TPM version and pass that in with the call. It is also possible that the caller could wrap the key for both TPMs and have the client decide which blob to use based on the TPM version. Because SRKs are statistically unique, there is a very small risk that this will accidentally provide a key that may be maliciously loaded on a TPM that it was not meant for.

```
HRESULT
TpmAttWrapPlatformKey(
    NCRYPT_KEY_HANDLE hInKey,
    BCRYPT_KEY_HANDLE hStorageKey,
    UINT32 tpmVersion,
    UINT32 keyUsage,
    _In_reads_opt_(cbPIN) PBYTE pbPIN,
    UINT32 cbPIN,
    UINT32 pcrMask,
    _In_reads_opt_(cbPcrTable) PBYTE pcrTable,
    UINT32 cbPcrTable,
    _Out_writes_to_opt_(cbOutput, *pcbResult) PBYTE pbOutput,
    UINT32 cbOutput,
    _Out_ PUINT32 pcbResult
    );
```

This API does not use the TPM. If the caller requests a TPM 1.2 key blob, this API will create a TPM_KEY12 structure with the BCRYPT_PCP_KEY_MAGIC magic prefix. If called for the TPM 2.0, a PCP_KEY_BLOB_WIN8 structure is generated. The key blobs are imported into the Platform Crypto Provider on the client with NCryptImportKey() as BCRYPT_OPAQUE_KEY_BLOB.

Parameters:

`hInKey` – NCrypt handle to an exportable key that the server intends to wrap.

`hStorageKey` – BCrypt public key handle of the SRK that will be used to wrap the key.

`tpmVersion` – The caller may select the target TPM version {TPM_VERSION_12, TPM_VERSION_20}.

keyUsage – Key usage policy that the key is restricted to. The default policy is NCRYPT_PCP_GENERIC_KEY. Please refer to NCRYPT_PCP_KEY_USAGE_POLICY:

```
NCRYPT_PCP_SIGNATURE_KEY = (0x00000001)
NCRYPT_PCP_ENCRYPTION_KEY = (0x00000002)
NCRYPT_PCP_GENERIC_KEY = (NCRYPT_PCP_SIGNATURE_KEY | NCRYPT_PCP_ENCRYPTION_KEY)
NCRYPT_PCP_STORAGE_KEY = (0x00000004)
```

PIN – String that is used as a PIN for the key. Please refer to the key property NCRYPT_PIN_PROPERTY.

pcrMask – The PCR mask to which a key is bound. PCR[0] corresponds to bit 0 up to bit 23 that corresponds to PCR[23].

pcrTable – Full set of all 24 SHA-1 PCRs that are used to calculate the PCR policy to which the key is supposed to adhere. This table has to be provided if a pcrMask is set.

Output, Result – Keyblob generated for the target TPM.

Overview of the PCP-Kit Package

The PCP-Kit package contains the following directories:

Directory	Contents
dll	Source for the PCP-Kit attestation functions described in the section "Attestation API Reference Implementation." Also low-level code to interface with TPM 1.2 or TPM 2.0.
exe	Source code for the PCPTool command-line utility (described in the section "PCPTool").
inc	Header files
misc	Batch files automating PCPTool to perform common actions and certificate template files to generate self-signed certificates with certreq.exe and a PFX file that may be used for certificate import.

The compressed sources should be extracted to a local directory on a development machine. The package contains the solution file. The libraries and executable are only useful on Windows 8 and above compatible machines with a TPM.

The remainder of this section describes the PCPTool command-line utility.

PCPTool

PCPTool is a command-line utility that can perform many of the steps needed by client and server-side attestation solutions. In source-code form it illustrates how BCrypt and the PCP-Kit libraries can be used to perform attestation-related TPM-functions and OS functions. In binary form it is a useful tool for administrators (or the curious) and can also be invoked by other applications.

PCP-Kit also contains a collection of batch files in the /misc directory. These scripts string together sequences of PCPTool operations to perform higher-level activities. For example, the AikCreation.cmd batch file automates the steps that need to be taken on a client to create a new AIK and "activate" it (obtain a new certificate). The scripts are mostly illustrative: for example, to actually create and activate an AIK, some of the commands would be performed on a server.

PCPTool is invoked from the command line. With no parameters, it generates the following help text:

```
PCPTool.exe
Microsoft PCPTool version 1.0 for Windows 8
Platform Integrity - TPM Attestation Reference Implementation.
Stefan Thom, stefanth@microsoft.com, 2011-2012
Copyright (c) Microsoft Corporation.  All rights reserved.

Commands:

General:
 GetVersion

RNG:
 GetRandom [size] {seed data} {output file}

Persistent TPM Keys:
 GetEK {key file}
 GetEKCert {cert file}
 AddEKCert [cert file]
 ExtractEK [cert file] {key file}
 GetSRK {key file}
 IssueEKCert [EKPub File] [Subject Name] {Cert file}

PCPKey Management:
 EnumerateKeys
 GetCertStore
 CreateKey [key name] {usageAuth | @ | ! } {migrationAuth} {pcrMask} {pcrs}
 ImportKey [key file] [key name] {usageAuth | @ | ! } {migrationAuth}
 ExportKey [key name] [migrationAuth] {key file}
 ChangeKeyUsageAuth [key name] [usageAuth] [newUsageAuth]
 DeleteKey [key name]
 GetPubKey [key name] {key File}
 Encrypt [pubkey file] [data] {blob file}
 Decrypt [key name] [blob file] {usageAuth}

AIK Management:
 CreateAIK [key name] {idBinding file} {nonce} {usageAuth | @ | ! }
 GetPubAIK [idBinding file] {key File}
 ChallengeAIK [idBinding file] [EKPub File] [secret] {Blob file} {nonce}
 ActivateAIK [key name] [Blob file]
 PrivacyCAChallenge [idBinding file] [EKPub File] [Subject] {Blob file} {nonce}
 PrivacyCAActivate [key name] [Blob file] {cert file}

Platform Configuration:
 GetPlatformCounters
 GetPCRs {pcrs file}
 GetLog [export file]
 GetArchivedLog [OsBootCount : @] [OsResumeCount : @] {export file}
 DecodeLog [log file]
 RegisterAIK [key name]
 EnumerateAIK

Platform Attestation:
 GetPlatformAttestation [aik name] {attestation file} {nonce} {aikAuth}
 CreatePlatformAttestationFromLog [log file] {attestation file} {aik name}
 DisplayPlatformAttestationFile [attestation file]
 ValidatePlatformAttestation [attestation file] [aikpub file] {nonce}

Key Attestation:
 GetKeyAttestation [key name] [aik name] {attest} {nonce} {keyAuth} {aikAuth}
 GetKeyAttestationFromKey [key name] {attest} {AIK name}
 ValidateKeyAttestation [attest] [aikpub file] {nonce} {pcrMask} {pcrs}
 GetKeyProperties [attest]

VSC Attestation:
 GetVscKeyAttestationFromKey {attest}

Key Hostage:
 WrapKey [cert Name] [storagePub file] {key file} {usageAuth} {pcrMask} {pcrs}
 ImportPlatformKey [key file] [key name] {cert file}
```

Commands

PCPTool is provided as sample code, and as such we do not provide full documentation of its options and capabilities. Additionally, PCPTool serves as a fairly thin wrapper over some advanced TPM capabilities—for example, the mechanisms for creating an AIK and obtaining a certificate. Readers will need to understand the behavior of the TPM for a full appreciation of the actions performed by PCPTool.

A slightly higher level of abstraction for TPM commands is in the scripts provided in the /misc directory of the PCP-Kit distribution. These scripts show how sequences of commands can be used to perform useful actions. For example, the BasicProviderTest.cmd script file demonstrates the creation, use, and management of TPM-protected keys.

Most PCPTool options take one or more command-line parameters. Many of the parameters are optional. PCPTool commands typically give results in XML to StdOut. Some also create output files with the name specified in the command line. The output files are typically raw binary (often the native TPM data structures). The binary files are not designed to support data interchange; they are merely a convenient way of piping partial results from one routine to another.

Several commands take optional nonces or usage-authorization data. The nonces and the actual TPM-key usage or migration authorization values are the (20-byte) SHA1 hash of the input string (such as performed by the ConvertToOwnerAuth WMI method).

PCPTool can only manipulate 2,048-bit RSA keys, regardless of the capabilities of the underlying TPM. However, the Platform Crypto Provider also supports 1,024-bit legacy keys.

Several commands create or manipulate keys in the `MS_PLATFORM_CRYPTO_PROVIDER` BCrypt provider. Keys in the key store are addressed by the alphanumeric name that they were given at creation. Keys in this provider can also have certificates associated with them.

Many of the actions need administrative privileges.

This section provides a brief overview of PCPTool command options. Sources are provided for more detailed understanding.

General

`GetVersion`

GetVersion returns the TPM version number and vendor information.

RNG

`GetRandom [size] {seed data} {output file}`

Gets random data from the TPM.

Persistent TPM Keys

`GetEK {key file}`

Returns the TPM EK public key and optionally saves it in a file.

`GetEKCert {cert file}`

Returns the copy of the TPM endorsement key certificate that is saved in MS_PLATFORM_CRYPTO_PROVIDER and optionally saves it in a file.

`AddEKCert [cert file]`

Registers the supplied certificate with the Endorsement Key in MS_PLATFORM_CRYPTO_PROVIDER.

`ExtractEK [cert file] {key file}`

Extracts the Endorsement public key from the supplied certificate file in a form that can be used by other PCPTool commands like PCPTool ChallengeAIK.

`GetSRK {key file}`

Returns the Storage Root Key in a form that can be used by other PCPTool commands.

`IssueEKCert [EKPub File] [Subject Name] {Cert file}`

Creates a simplified EK certificate. This certificate may be added to the EKCert store with the command AddEKCert. The certificate will be signed with a CA certificate. If there are multiple CA certs on the machine, the user will be able to select the cert to use. The CA cert may have been created by a child CA of an Enterprise CA or as a self-signed cert with 'certreq –new cacert.inf cacert.cer' where cacert.inf has the following content:

```
[NewRequest]
   Subject = "CN=TPM Endorsement CA01"
   HashAlgorithm = sha256
   KeyAlgorithm = RSA
   KeyLength = 2048
   KeyUsage = "CERT_DIGITAL_SIGNATURE_KEY_USAGE | CERT_KEY_CERT_SIGN_KEY_USAGE |
CERT_CRL_SIGN_KEY_USAGE"
   KeyUsageProperty = "NCRYPT_ALLOW_SIGNING_FLAG"
   ProviderName = "Microsoft Software Key Storage Provider"
   RequestType = Cert
   Exportable = true
   ExportableEncrypted = false
```

PCPKey Management

`EnumerateKeys`

Returns the public properties of all TPM keys in MS_PLATFORM_CRYPTO_PROVIDER. The tool must run with elevated permissions to enumerate machine keys.

`GetCertStore`

Creates a memory cert store that contains all PCPKSP stored certificates that are in the purview of the current user.

`CreateKey [key name] {usageAuth | @ | ! } {migrationAuth} {pcrMask} {PCRs}`

Creates a new TPM key that is a child of the Storage Root Key. By default the new key has the parameters listed below. Optional parameters allow a PIN to be set. If PCR configuration parameters are supplied, then the key is accessible only when the PCRs are set as indicated. pcrMask is a bitmap represented as an integer where bit 0 corresponds with PCR[0] through bit 23 that corresponds to PCR[23] (Bit 24-32 are ignored). The parameter PCRs is a file containing 24 SHA-1 digests of the PCR values (only the selected PCRs are used for key authorization) to be used by this operation. The file format of PCRs is binary such as that produced by PCPTool GetPcrs.

Default key parameters are:

```
<Key>
    <Algorithm>RSA</Algorithm>
    <MachineKey>FALSE</MachineKey>
    <Name>k0</Name>
    <PubKeyDigest>6445d60671f6ef216056717845b33dfd7e4f67bf</PubKeyDigest>
    <KeyLength>2048</KeyLength>
    <KeyUsage>GENERIC</KeyUsage>
    <PINRequired>FALSE</PINRequired>
    <ExportAllowed>FALSE</ExportAllowed>
```

The *usageAuth* may be provided through the Windows UI if the user sets the at-sign (@) symbol or the exclamation point (!) symbol as *usageAuth*. Using the at-sign (@) lets the user choose whether the key should be created with just a consent request on every use or with a PIN. Using the exclamation point (!) requires the use of a PIN with the key. The UI presented below is the consent/optional PIN dialog:

`ImportKey [key file] [key name] {usageAuth | @ | ! } {migrationAuth}`

Import an RSA key pair into the TPM and key store. The parameter [key file] is a BCRYPT_RSAKEY_BLOB as it is created by PCPTool ExportKey. The *usageAuth* may be provided through the Windows UI if the user sets the at-sign (@) symbol or exclamation point (!) as *usageAuth*. Using the at-sign (@) symbol lets the user choose whether the key should be created with just a consent request on every use or with a PIN. Using the exclamation point (!) requires the use of a PIN with the key.

`ExportKey [key name] [migrationAuth] {key file}`

Export an exportable key from the key store and TPM. The caller must provide valid migration authorization. The key is exported in binary form as a BCRYPT_RSAKEY_BLOB. An example of the exported key format is provided here:

```
<RSAKey size="539" keyName=”k0”>
  <Magic>RSA2<!-- 0x32415352 --></Magic>
  <BitLength>2048</BitLength>
  <PublicExp size="3">
    010001
  </PublicExp>
  <Modulus size="256" digest=”6445d60671f6ef216056717845b33dfd7e4f67bf”>
    c2fc224f6cb52116adc4f58201cdcc26e38ff7c60a76e3ed543606c508fe99bb76ef8625f4
ad5640e4d131981d5b9cc8a610f87c47340fcf83399b110f96c7239dd31ce3a45dd74ab30dc5598f
8abfa3b31d5e27c109cb5bff45f2c2f49637197a2d0fec119b95566e24b9954d51aaad01d0fa88a6
0075505be35d6fb949fd892a48c5fb69956d1662db67c9d4fc9ba0c39c64d39528d616f12bfa39a9
5df5f25edbcfdf245ce4355086d2a5a0b48dcb13ab63f68498054875fcc07762a7f2b47e45597853
484a03ee0789e9a79a4c8f212e23e5b6a25ee31fb821e99d3acede5a03efadc47fba404076fb7826
aeaadb846fe2772ff8a1dbaec67e0661cdf32b
  </Modulus>
  <Prime1 size="128">
    df5c86dc252824d83cb4d1330f3dfe3bc5ac776db3545079181cdbb9216ac739f309abffa0
8d7a90acf8fa5248d108222e0e4c5b1112bbdc7110353b3282996d76425c5b2c81d6d80dedf56c74
8a97ed5f3ea887583d060ab7f2c7da574b4326133ff954b8c06f476a28ebf4887618638b37b40c46
64273556e014133f5f6e59
  </Prime1>
  <Prime2 size="128">
    df7a1a10091bface5825611c7da0aa137393f62b47a62bb3d7e8d457c61bc0e6ab38ab6abb
5af95b8865bd2270c9891ca0923088238ce51214b7847f9030a2e945a4b72e00b307aea371f15b09
2794ad026007ce742482817d534e97169aae674af32e305654388385a88f6cf886c10b954d908b50
f84422b230eb8fdf672523
  </Prime2>
</RSAKey>
```

`ChangeKeyUsageAuth [key name] [usageAuth] [newUsageAuth]`

Change the use-authorization of the named key. The caller must provide the current use-authorization to perform this command.

`DeleteKey [key name]`

Delete the named key from the key store.

`GetPubKey [key name] {key File}`

Return the public part of the named key as a BCRYPT_RSAKEY_BLOB. Please note that this key has the magic RSA1 and no primes are set.

```
<RSAKey size="539" keyName=”k0”>
  <Magic>RSA1<!-- 0x31415352 --></Magic>
  <BitLength>2048</BitLength>
  <PublicExp size="3">
    010001
  </PublicExp>
  <Modulus size="256" digest=”6445d60671f6ef216056717845b33dfd7e4f67bf”>
    c2fc224f6cb52116adc4f58201cdcc26e38ff7c60a76e3ed543606c508fe99bb76ef8625f4
ad5640e4d131981d5b9cc8a610f87c47340fcf83399b110f96c7239dd31ce3a45dd74ab30dc5598f
8abfa3b31d5e27c109cb5bff45f2c2f49637197a2d0fec119b95566e24b9954d51aaad01d0fa88a6
0075505be35d6fb949fd892a48c5fb69956d1662db67c9d4fc9ba0c39c64d39528d616f12bfa39a9
5df5f25edbcfdf245ce4355086d2a5a0b48dcb13ab63f68498054875fcc07762a7f2b47e45597853
484a03ee0789e9a79a4c8f212e23e5b6a25ee31fb821e99d3acede5a03efadc47fba404076fb7826
aeaadb846fe2772ff8a1dbaec67e0661cdf32b
  </Modulus>
  <Prime1/>
  <Prime2/>
</RSAKey>
```

`Encrypt [pubkey file] [data] {blob file}`

Encrypt data using the public key provided in the pubkey file. The pubkey file can be created using the GetPubKey command.

`Decrypt [key name] [blob file] {usageAuth}`

Decrypt with the named key data that was encrypted using the Encrypt command above. If the key requires authorization or consent and no password is provided, Windows will display the UI. Below is the UI for a key that requires consent:

AIK Management

The commands in this section perform the actions that are needed to create an Attestation Identity Key (AIK) and obtain a certificate for it. In the following description, the *client* is the entity with a TPM that is requesting an AIK certificate, and the *server* is receiving the request.

One possible work-flow is given below (more complex standard protocols are described in TCG specifications).

Commands / Actions	Client/ Server	Comments
PCPTool CreateAIK aik0 binding_0	C	Creates a new AIK and assigns it the name aik0. The "binding" contains information about the freshly created AIK, including the public key.
PCPTool GetEK EK	C	Set the file EK to the Endorsement Key of the client system.

Client sends binding_0, EK (and possibly Endorsement Key certificates) to the server. The server examines the EK and certificate to determine whether the client system is trustworthy. If deemed trustworthy, then the server performs these steps:

PCPTool GetPubAIK binding_0 aik0_pub	S	Extract the public key of the freshly created AIK from the binding file.

The server creates a certificate over the public key of the AIK. The certificate can contain any information relevant for later security assessment. The server then creates a random key-string ("asdfqwer" in this example) and encrypts the certificate using this key and a cipher of its choosing.

PCPTool ChallengeAIK binding_0 ek asdfqwer encrypted_secret	S	ChallengeAIK encrypts the secret "asdfqwer" using the TPM Endorsement Key in a form that can be retrieved by TPM_ActivateIdentity or TPM2_ActivateCredential. The encrypted blob also contains instructions to the recipient TPM to reveal the encrypted secret only if the AIK named in binding_0 is loaded on the recipient device.

The server sends encrypted_secret and the certificate that was encrypted with the secret "asdfqwer" to the client

PCPTool ActivateAIK aik0 encrypted_secret	C	This command asks the TPM to decrypt encrypted_secret with the EK using the command TPM_ActivateIdentity or TPM2_ActivateCredential. The TPM will do this only if there is a loaded AIK with the public key that was specified in the ChallengeAIK step. If this command succeeds, the following output is generated: <Activation> <Secret size="18">asdfqwer</Secret> </Activation>

The client can now decrypt the encrypted AIK certificate using the key that was revealed in ActivateIdentity. At this time the key can be used to generate quotes (or other key or platform attestations, as described in this paper).

PCPTool RegisterAIK	C	(Optional) Register the AIK for use in signing the boot log.

The commands themselves are as follows:

`CreateAIK [key name] {idBinding file} {nonce} {usageAuth | @ | ! }`

Create an AIK and assign it the name provided. The output (typically stored in the binding file) contains the public key of the AIK. The *usageAuth* may be provided through the Windows UI if the user sets the at-sign (@) symbol or the exclamation point (!) as *usageAuth*. Using the at-sign

(@) symbol lets the user choose whether the key should be created with just a consent request on every use or with a PIN. Using the exclamation point (!) requires use of a PIN with the key.

`GetPubAIK [idBinding file] {key File}`

Extract the AIK public key from the binding file. It produces the same output as PCPTool GetPubKey. This is typically run on a server to generate a certificate with the AIK_{PUB}.

`ChallengeAIK [idBinding file] [EKPub File] [secret] {Blob file} {nonce}`

Encrypt the string "secret" with EKPub and associate it with the AIK-public key in the binding file. This uses TCG AIK-activation protocols and data structures. This is typically run on a server.

`ActivateAIK [key name] [Blob file]`

Passes the blob file to the TPM_ActivateIdentity or TPM2_ActivateCredential commands on the underlying TPM. If the command succeeds, the secret provided to ChallengeAIK is returned.

`PrivacyCAChallenge [idBinding file] [EKPub File] [Subject] {Blob file} {nonce}`

This command is a PrivacyCA function analog to ChallengeAIK. It will generate an AIK certificate and sign the certificate with the CA keys on the machine. The certificate is encrypted with a random AES128 key that is then encrypted in the activation blob. The creation of a CA certificate is detailed in the section "Persistent TPM Keys."

`PrivacyCAActivate [key name] [Blob file] {cert file}`

This command is the client-side PrivacyCA function analog to ActivateAIK. It will activate the AIK and then unwrap the AIK certificate. If successful, the AIK certificate is added to the user's 'My' store and as a property on the PCP key.

Platform Configuration

`GetPlatformCounters`

Lists boot and associated event-counters and their current values.

`GetPCRs {pcrs file}`

Gets all 24 current PCR values from the TPM.

`GetLog [export file]`

Retrieves the current boot log in XML form (and binary form if an export file is provided). The XML form also calculates the PCR values from the events in the log. Log consistency can be checked with the command option ValidatePlatformAttestation.

`GetArchivedLog [OsBootCount : @] [OsResumeCount : @] {export file}`

Returns a log from a prior boot or resume event.

`DecodeLog [log file]`

Decodes a binary-log file created by GetLog into XML.

`RegisterAIK [key name]`

This function will register an AIK so it will be used to generate a trust point in the WBCL log every time the system is booted or resumed from hibernation and certify all non-exportable keys that are created in the TPM. Note that this key may not require authorization. The registry value used to register the key will be set to the key name. Also note that each registered AIK will

occupy the TPM at the crucial startup or resume time and may have an effect on system performance.

`EnumerateAIK`

This command will list all AIKs registered to produce trust points and key certifications.

Platform Attestation

Commands in this section generate and manipulate platform attestations. For example, options enable signing (quoting) of PCRs and validation that English-readable logs are consistent with stated PCR values.

`GetPlatformAttestation [aik name] {attestation file} {nonce} {aikAuth}`

Get a platform attestation (a quote using the named AIK over all current PCR values, together with the current boot log). If an attestation file parameter is provided, the attestation is saved in a binary file.

`CreatePlatformAttestationFromLog [log file] {attestation file} {aik name}`

This function uses a trust point that has to be present in the WBCL to turn it into an attestation structure, so it may be validated by ValidatePlatformAttestation.

`DisplayPlatformAttestationFile [attestation file]`

Translates and displays a binary platform attestation file in XML.

`ValidatePlatformAttestation [attestation file] [aikpub file] {nonce}`

This is a server-side function that validates that an attestation is well-formed. Specifically, the following steps are performed:

1. Validates that the signature over quoted data is well-formed and can be verified with the public key in AIK_{PUB}.
2. Checks that the attestation structure is well-formed and that the nonce matches the provided one.
3. Validates that the PCR registers and values in the quote are those described in the attestation file.
4. Validates that the PCR values in the attestation file match those quoted.
5. Validates that the log entries in the attestation file match the PCR entries in the attestation file.

Typically, a caller will perform additional tests (checking that the AIK is trustworthy, for example).

Key Attestation

Key attestation is a TPM mechanism that allows a TPM AIK to certify the properties of another loaded key. This allows a remote entity to validate that a key is actually protected by a TPM.

`GetKeyAttestation [key name] [aik name] {attest} {nonce} {keyAuth} {aikAuth}`

Uses the named AIK to generate a certificate for another key.

`GetKeyAttestationFromKey [key name] {attest} {AIK name}`

Retrieves the automatic key attestation data from the named key. The AIK name allows the user to select which data blob to retrieve. This function does not perform the actual key attestation. It only retrieves the attestation that was generated at key creation time with the registered AIKs.

`ValidateKeyAttestation [attest] [aikpub file] {nonce} {pcrMask} {pcrs}`

This is a server-side operation to validate that a key attestation generated by GetKeyAttestation is a properly formed attestation structure from the AIK with the public key provided.

`GetKeyProperties [attest]`

This is a server-side operation that returns the properties of the key described in the attest file.

Virtual Smart Card Attestation

`GetVscKeyAttestationFromKey {attest} {AIK name}`

This function allows the retrieval of the automatic key attestation data for a key that was created on a virtual smart card (VSC) analog to GetKeyAttestationFromKey. For this function to succeed, the caller has to be running as elevated administrator, local service or network service. This function will first show smart card selector UI and then allow the user to pick a certificate from the smart card. The command will then open the PCP local system key storage location and browse for the public key that matches the one from the certificate. Since VSC runs as local service, all keys that are created on smart cards are created in that space. Once the key is found, it will be opened and the attestation information will be exported. However, the VSC key will not be usable by PCPTool while open because its internal authorization value is not present.

Key Hostage

`WrapKey [cert Name] [storagePub file] {key file} {usageAuth} {pcrMask} {pcrs}`

This is a server-side operation that creates a key that can be loaded on a TPM with the storage key described by cert_name and storage_pub. The key can be optionally bound to PCR values. This key will only be loadable on the associated TPM when the PCRs are set appropriately.

`ImportPlatformKey [key file] [key name] {cert file}`

Imports a key created with WrapKey so that it can be used on the TPM. Optionally, a provided certificate for that key will be imported with the key and registered in the user's certificate store.

Scenario Scripts

The provided scripts give some insight into how the commands are used together to go through a scenario. These scripts may also be used as a quick build verification and/or platform compliance test.

BasicProviderTest.cmd

Basic Provider test will run the following calls in this sequence to create and use non-authorized and authorized keys. In addition, the test will change PINs and perform authorized key export and import.

```
1. PCPTool GetVersion
2. PCPTool GetRandom 1024 ThisIsASeedForTheRNGInTheTPM
3. PCPTool GetSRK
4. PCPTool CreateKey pcptestkey1
5. PCPTool CreateKey pcptestkey2 MySuperSecretUsagePIN
```

```
6. PCPTool CreateKey pcptestkey3 MySuperSecretUsagePIN TheAdministratorsPIN
7. PCPTool EnumerateKeys
8. PCPTool GetPubKey {pcptestkey1, pcptestkey2, pcptestkey3} {pcptestkey1Pub,
   pcptestkey2Pub, pcptestkey3Pub}
9. PCPTool Encrypt {pcptestkey1Pub, pcptestkey2Pub, pcptestkey3Pub}
   SuperSecretSecret {pcptestkey1Blob, pcptestkey2Blob, pcptestkey3Blob}
10. PCPTool Decrypt pcptestkey1 pcptestkey1Blob
11. PCPTool Decrypt pcptestkey2 pcptestkey2Blob MySuperSecretUsagePIN
12. PCPTool Decrypt pcptestkey3 pcptestkey3Blob MySuperSecretUsagePIN
13. PCPTool ChangeKeyUsageAuth {pcptestkey1, pcptestkey2} MySuperSecretUsagePIN
    MyOtherSuperSecretUsagePIN
14. PCPTool Decrypt pcptestkey2 pcptestkey2Blob MyOtherSuperSecretUsagePIN
15. PCPTool Decrypt pcptestkey3 pcptestkey3Blob MyOtherSuperSecretUsagePIN
16. PCPTool ExportKey pcptestkey3 TheAdministratorsPIN pcptestkey3
17. PCPTool DeleteKey {pcptestkey1, pcptestkey2, pcptestkey3}
18. PCPTool ImportKey pcptestkey3 pcptestkey1
19. PCPTool ImportKey pcptestkey3 pcptestkey2 MySuperSecretUsagePIN
20. PCPTool ImportKey pcptestkey3 pcptestkey3 MySuperSecretUsagePIN
    TheAdministratorsPIN
21. PCPTool Decrypt pcptestkey1 pcptestkey3Blob
22. PCPTool Decrypt pcptestkey2 pcptestkey3Blob MySuperSecretUsagePIN
23. PCPTool Decrypt pcptestkey2 pcptestkey3Blob MySuperSecretUsagePIN
24. PCPTool DeleteKey {pcptestkey1, pcptestkey2, pcptestkey3}
```

PCRBoundKeyTest.cmd

This test will create three keys bound to PCR values from three different sources. The first one binds to the current state; the second one binds to the state that was previously recorded from the TPM; and the third binds with an entirely random set of PCRs. The keys are then used to show that PCR binding of a key works properly.

```
1. PCPTool GetVersion
2. PCPTool GetPCRs goodPcrs
3. PCPTool GetRandom 480 "" badPcrs
4. PCPTool CreateKey pcptestkey1 "" "" 0x0000ffff → Current PCR measurements from
   TPM
5. PCPTool CreateKey pcptestkey2 "" "" 0x0000ffff goodPcrs  →  Recorded PCRs
   Measurements
6. PCPTool CreateKey pcptestkey3 "" "" 0x0000ffff badPcrs  →  Random PCR values
7. PCPTool EnumerateKeys
8. PCPTool GetPubKey {pcptestkey1, pcptestkey2, pcptestkey3} {pcptestkey1Pub,
   pcptestkey2Pub, pcptestkey3Pub}
9. PCPTool Encrypt {pcptestkey1Pub, pcptestkey2Pub, pcptestkey3Pub}
   SuperSecretSecret {pcptestkey1Blob, pcptestkey2Blob, pcptestkey3Blob}
10. PCPTool Decrypt {pcptestkey1, pcptestkey2, pcptestkey3} {pcptestkey1Blob,
    pcptestkey2Blob, pcptestkey3Blob}
11. PCPTool Decrypt pcptestkey3 pcptestkey3Blob  → Operation has to fail
12. PCPTool DeleteKey {pcptestkey1, pcptestkey2, pcptestkey3}
```

PrivacyCA.cmd

This script creates a sample enterprise EKCert and adds it to the EKCertStore.

```
1. PCPTool GetVersion
2. Generate a CACert
3. PCPTool GetEK EKpub
4. PCPTool IssueEKCert EKpub EnterpriseTPM EnterpriseEKCert.cer
5. PCPTool AddEKCert EnterpriseEKCert.cer
6. PCPTool CreateAIK pcptestAIK idBinding NonceFromTheServerForKeyCreation
7. PCPTool GetPubAIK idBinding Aikpub
8. PCPTool PrivacyCaChallenge idBinding EKpub EnterpriseAIK activationBlob
   NonceFromTheServerForKeyCreation
```

```
9.  PCPTool PrivacyCaActivate EnterpriseAIK activationBlob AIKCert.cer
10. PCPTool EnumerateKeys
```

AikCreation.cmd

This script shows how the AIK handshake may be performed and how to register an AIK to create the trust points. In order to run TrustPointValidation.cmd successfully, the system has to be rebooted after this script is run.

```
1.  PCPTool GetVersion
2.  PCPTool GetEKCert EKCert   → If EK_Cert is available
3.  PCPTool ExtractEK EKCert EKpub
4.  PCPTool GetEK EKpub   → Alternate way to obtain EK_PUB
5.  PCPTool CreateAIK pcptestAIK idBinding NonceFromTheServerForKeyCreation
6.  PCPTool GetPubAIK idBinding Aikpub
7.  PCPTool ChallengeAIK idBinding EKpub SecretNonceFromServer activationBlob
    NonceFromTheServerForKeyCreation
8.  PCPTool ActivateAIK pcptestAIK activationBlob
9.  PCPTool EnumerateKeys
10. PCPTool GetPubKey pcptestAIK Aikpub
11. PCPTool RegisterAIK pcptestAIK
12. PCPTool EnumerateAIK
13. bcdedit -set {globalsettings} integrityservices enable
```

Attestation.cmd

This script will create an unsigned and a signed platform attestation with a nonce and a previously created AIK. The successful run of this script requires a prior successful run of AikCreation.cmd.

```
1.  PCPTool GetVersion
2.  PCPTool EnumerateKeys
3.  PCPTool GetPubKey pcptestAIK Aikpub
4.  PCPTool GetPlatformCounters
5.  PCPTool GetPlatformAttestation pcptestAIK attestationBlob
    ThisIsANonceProvidedFromTheServer
6.  PCPTool ValidatePlatformAttestation attestationBlob Aikpub
    ThisIsANonceProvidedFromTheServer
7.  PCPTool ValidatePlatformAttestation attestationBlob ""
    ThisIsANonceProvidedFromTheServer
8.  PCPTool DisplayPlatformAttestationFile attestationBlob
9.  PCPTool GetPlatformAttestation "" localAttestationBlob
    ThisIsANonceProvidedFromTheServer   → Unsigned Attestation because no key name
10. PCPTool ValidatePlatformAttestation localAttestationBlob ""
    ThisIsANonceProvidedFromTheServer
11. PCPTool DisplayPlatformAttestationFile localAttestationBlob
```

TrustPointValidation.cmd

In order to run this test, AikCreation.cmd has to have run successfully and the machine has to be rebooted. It will use the trust point attestation in the log for integrity validation.

```
1.  PCPTool GetVersion
2.  PCPTool EnumerateAIK
3.  PCPTool GetPubKey pcptestAIK Aikpub
4.  PCPTool GetPlatformCounters
5.  PCPTool GetLog currentLog
6.  PCPTool CreatePlatformAttestationFromLog currentLog currentAttestation
    pcptestAIK
7.  PCPTool ValidatePlatformAttestation currentAttestation Aikpub
8.  PCPTool ValidatePlatformAttestation currentAttestation ""   → Validation to PCRs
9.  PCPTool DisplayPlatformAttestationFile currentAttestation
10. PCPTool GetArchivedLog @ 0 lastBootLog   → Get the initial boot log of the cycle
```

```
11. PCPTool CreatePlatformAttestationFromLog lastBootLog lastBootAttestation
    pcptestAIK
12. PCPTool ValidatePlatformAttestation lastBootAttestation Aikpub
13. PCPTool DisplayPlatformAttestationFile lastBootAttestation
```

KeyAttestationTest.cmd

This script creates three keys with different usage policies—regular, PIN-protected, and PCR-bound—and attests them with the AIK. In order to run this script, AikCreation.cmd has to have run successfully.

```
1.  PCPTool GetVersion
2.  PCPTool GetPCRs goodPcrs
3.  PCPTool CreateKey pcptestkey1
4.  PCPTool CreateKey pcptestkey2 MySuperSecretUsagePIN
5.  PCPTool CreateKey pcptestkey3 "" "" 0x0000ffff goodPcrs
6.  PCPTool EnumerateKeys
7.  PCPTool GetPubKey pcptestAIK Aikpub
8.  PCPTool GetKeyAttestation pcptestkey1 pcptestAIK pcptestkey1Attest
    ThisIsANonceProvidedFromTheServer
9.  PCPTool GetKeyAttestation pcptestkey2 pcptestAIK pcptestkey2Attest
    ThisIsANonceProvidedFromTheServer MySuperSecretUsagePIN
10. PCPTool GetKeyAttestation pcptestkey3 pcptestAIK pcptestkey3Attest
    ThisIsANonceProvidedFromTheServer
11. PCPTool ValidateKeyAttestation {pcptestkey1Attest, pcptestkey2Attest,
    pcptestkey3Attest} Aikpub ThisIsANonceProvidedFromTheServer
12. PCPTool ValidateKeyAttestation pcptestkey3Attest Aikpub
    ThisIsANonceProvidedFromTheServer 0x0000ffff goodPcrs  → Also validate the PCRs
13. PCPTool GetKeyProperties {pcptestkey1Attest, pcptestkey2Attest,
    pcptestkey3Attest}
14. PCPTool DeleteKey {pcptestkey1, pcptestkey2, pcptestkey3}
```

AutoKeyAttestValidation.cmd

This script creates three keys with different usage policies—regular, PIN-protected, and PCR-bound—and retrieves the auto attestation data that was made with the registered AIK. In order to run this script, AikCreation.cmd has to have run successfully.

```
1.  PCPTool GetVersion
2.  PCPTool GetPCRs goodPcrs
3.  PCPTool CreateKey pcptestkey1
4.  PCPTool CreateKey pcptestkey2 MySuperSecretUsagePIN
5.  PCPTool CreateKey pcptestkey3 "" "" 0x0000ffff goodPcrs
6.  PCPTool EnumerateKeys
7.  PCPTool GetPubKey pcptestAIK Aikpub
8.  PCPTool GetKeyAttestationFromKey pcptestkey1 pcptestkey1Attest
9.  PCPTool GetKeyAttestationFromKey pcptestkey2 pcptestkey2Attest
10. PCPTool GetKeyAttestationFromKey pcptestkey3 pcptestkey3Attest
11. PCPTool ValidateKeyAttestation {pcptestkey1Attest, pcptestkey2Attest,
    pcptestkey3Attest} Aikpub
12. PCPTool ValidateKeyAttestation pcptestkey3Attest Aikpub
    ThisIsANonceProvidedFromTheServer 0x0000ffff goodPcrs  → Also validate the PCRs
13. PCPTool GetKeyProperties {pcptestkey1Attest, pcptestkey2Attest,
    pcptestkey3Attest}
14. PCPTool DeleteKey {pcptestkey1, pcptestkey2, pcptestkey3}
```

HostageKey.cmd

This script creates a self-signed certificate in the software KSP and exports the private key to be wrapped for the TPM with different policies. The keys and certificate are imported in the Platform Crypto Provider and tested.

```
1.  PCPTool GetVersion
2.  CertReq -new -binary -f CertTemplate.inf Hostage.Cer
3.  PCPTool GetSRK SRKpub
4.  PCPTool GetPCRs Pcrs
5.  PCPTool WrapKey Hostage.Cer SRKpub Hostage
6.  PCPTool ImportPlatformKey Hostage Hostage Hostage.Cer
7.  PCPTool GetPubKey Hostage HostagePub
8.  PCPTool Encrypt HostagePub SuperSecretSecret SecretBlob
9.  PCPTool Decrypt Hostage SecretBlob
10. PCPTool DeleteKey Hostage
11. PCPTool WrapKey Hostage.Cer SRKpub Hostage MySuperSecretUsagePIN
12. PCPTool ImportPlatformKey Hostage Hostage Hostage.Cer
13. PCPTool Encrypt HostagePub SuperSecretSecret SecretBlob
14. PCPTool Decrypt Hostage SecretBlob MySuperSecretUsagePIN
15. PCPTool DeleteKey Hostage
16. PCPTool WrapKey Hostage.Cer SRKpub Hostage "" 0x0000ffff Pcrs
17. PCPTool ImportPlatformKey Hostage Hostage Hostage.Cer
18. PCPTool Encrypt HostagePub SuperSecretSecret SecretBlob
19. PCPTool Decrypt Hostage SecretBlob
20. PCPTool DeleteKey Hostage
21. PCPTool WrapKey Hostage.Cer SRKpub Hostage MySuperSecretUsagePIN 0x0000ffff
    Pcrs
22. PCPTool ImportPlatformKey Hostage Hostage Hostage.Cer
23. PCPTool Encrypt HostagePub SuperSecretSecret SecretBlob
24. PCPTool Decrypt Hostage SecretBlob MySuperSecretUsagePIN
25. PCPTool DeleteKey Hostage
```

Certificate Enrollment Templates

The distribution also provides certificate templates for use with certreq.exe to generate a self-signed certificate that may be used in other certificate-based Windows applications.

The certificate is created with the in-box utility certreq.exe:

```
CertReq -new PCPCert.inf PCPCert.cer
```

PCPCert.inf

This template will create a legacy platform-bound RSA 1,024-bit signing certificate.

```
[NewRequest]
    Subject = "CN=WeakPCPTestCert"
    HashAlgorithm = sha1
    KeyAlgorithm = RSA
    KeyLength = 1024
    KeyUsage = "CERT_DIGITAL_SIGNATURE_KEY_USAGE"
    KeyUsageProperty = "NCRYPT_ALLOW_SIGNING_FLAG"
    ProviderName = "Microsoft Platform Crypto Provider"
    RequestType = Cert
    FriendlyName = "DeleteMe!"
    Exportable = false
    ExportableEncrypted = false

[EnhancedKeyUsageExtension]
    OID=2.5.29.37.0
```

PCPCertPIN.inf

This template will create a strong platform-bound RSA 2,048-bit signing certificate, with key usage consent, and the user may set a PIN on the key at creation time. If the KeyProtection option is changed to NCRYPT_UI_FORCE_HIGH_PROTECTION_FLAG, the optional PIN becomes a mandatory PIN.

```
[NewRequest]
    Subject = "CN=StrongPCPTestCert"
    HashAlgorithm = sha256
    KeyAlgorithm = RSA
    KeyLength = 2048
    KeyUsage = "CERT_DIGITAL_SIGNATURE_KEY_USAGE"
    KeyUsageProperty = "NCRYPT_ALLOW_SIGNING_FLAG"
    ProviderName = "Microsoft Platform Crypto Provider"
    RequestType = Cert
    KeyProtection = NCRYPT_UI_PROTECT_KEY_FLAG
#    KeyProtection = NCRYPT_UI_FORCE_HIGH_PROTECTION_FLAG
    FriendlyName = "DeleteMe!"
    Exportable = false
    ExportableEncrypted = false

[EnhancedKeyUsageExtension]
    OID=2.5.29.37.0
```

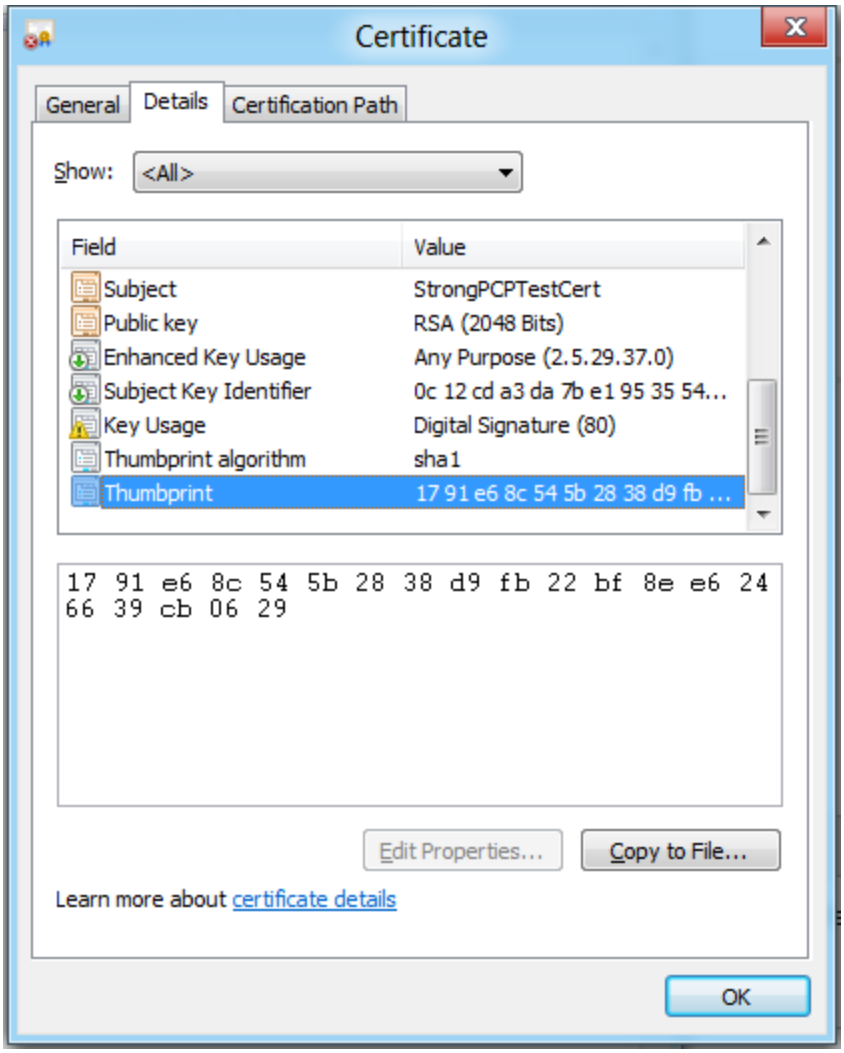
CACert.inf

This template will create a CA Certificate that is required for the PrivacyCA functions.

```
[NewRequest]
   Subject = "CN=TPM Endorsement CA01"
   HashAlgorithm = sha256
   KeyAlgorithm = RSA
   KeyLength = 2048
   KeyUsage = "CERT_DIGITAL_SIGNATURE_KEY_USAGE |
CERT_KEY_CERT_SIGN_KEY_USAGE | CERT_CRL_SIGN_KEY_USAGE"
   KeyUsageProperty = "NCRYPT_ALLOW_SIGNING_FLAG"
   ProviderName = "Microsoft Software Key Storage Provider"
   RequestType = Cert
   Exportable = true
   ExportableEncrypted = false
```

Testing a Self-Signed Certificate

The created self-signed certificate can be tested with the utility CertUtil.exe. To do this, open the created certificate PCPCert.cer or look it up in CertMgr.msc and get the thumbprint of the certificate.


Certificate
General
Details
Certification Path
Show:
<All>
Field
Value
Subject
StrongPCPTestCert
Public key
RSA (2048 Bits)
Enhanced Key Usage
Any Purpose (2.5.29.37.0)
Subject Key Identifier
0c 12 cd a3 da 7b e1 95 35 54...
Key Usage
Digital Signature (80)
Thumbprint algorithm
sha1
Thumbprint
17 91 e6 8c 54 5b 28 38 d9 fb ...
17 91 e6 8c 54 5b 28 38 d9 fb 22 bf 8e e6 24
66 39 cb 06 29
Edit Properties...
Copy to File...
Learn more about certificate details
OK

Then, call CertUtil.exe with that thumbprint. CertUtil will show the authorization UI if the key requires it:

```
C:\Windows\system32>certutil -store -user my 1791e68c545b2838d9fb22bf8ee6246639cb0629
my "Personal"
================ Certificate 6 ================
Serial Number: 6681809e6abc3b8447cfa4d3cab66581
Issuer: CN=StrongPCPTestCert
 NotBefore: 11/4/2011 12:02 PM
 NotAfter: 11/4/2012 11:22 AM
Subject: CN=StrongPCPTestCert
Signature matches Public Key
Root Certificate: Subject matches Issuer
Cert Hash(sha1): 17 91 e6 8c 54 5b 28 38 d9 fb 22 bf 8e e6 24 66 39 cb 06 29
  Key Container = CertReq-1bd0f964-992a-41dd-ba5a-a4218cedb0de
  Unique container name: C:\Users\XXXX\AppData\Local\Microsoft\Crypt
o\PCPKSP\8aefa5d20304c73d63c74ad7a7297e6ee110ca94\d3b6cdf5cac273f9bbb910e4bee996
b1b3028a0b.PCPKEY
  Provider = Microsoft Platform Crypto Provider
Encryption test passed
CertUtil: -store command completed successfully.
```

PFX Private Key and Certificate Import

In the /misc directory is a HostageCert.pfx file (password: zaqwsx) that contains a key pair and certificate that may be used to test PFX import with the in-box utility CertUtil.exe. PFX files are widely used to provision encryption certificates to specific systems.

To ensure that the provided key pair cannot be distributed any further by an authorized user of the key, the administrator will import the PFX file into the Platform Crypto Provider. The key will appear exportable; however, the *migrationAuth* is an unknown long random number that is hard to guess and the key may be seen as not exportable.

HostageCert.pfx

The command for the import of the given PFX file is:

```
CertUtil.exe -f -v -p zaqwsx -user -csp "Microsoft Platform Crypto
Provider" -importpfx HostageCert.pfx NoExport
```

After the import, the certificate will show up in the user's MY store in CertMgr.msc.

Windows Attestation Scenarios

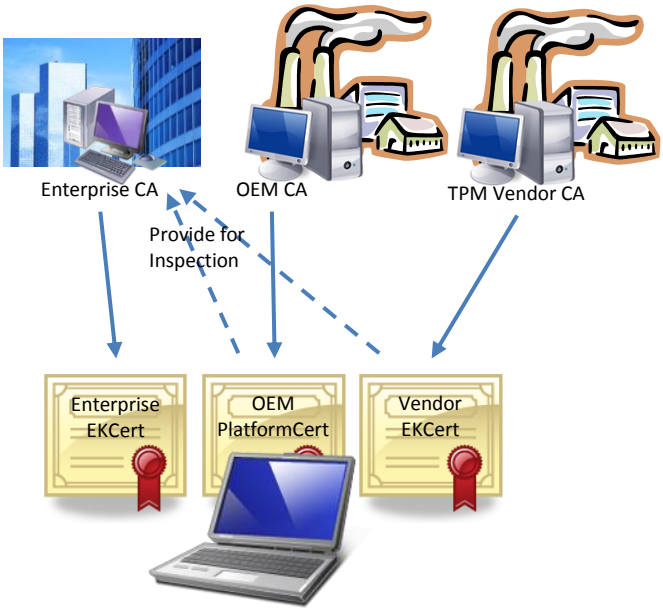
The new TPM provisioning, the Platform Crypto Provider, and the Attestation Reference Implementation are new platform capabilities that enable whole new classes of applications to be developed for Windows operating systems. This section describes some scenarios that may be realized by third-party tools and utilities.

A key aspect of the following scenarios is to lower the cost of ownership by removing the need for an administrator to be physically present at a machine in order to gain trust in it or its configuration. These scenarios drive user self-service, while upholding policy enforcement for the enterprise administrator.

It is important to note that all platform configuration attestation requires that the local administrator of the machine have a vested interest in the scenario being secure.

The administrator caveats do not apply to key attestation, because key attestation involves only the TPM itself. In this case a TPM manufacturer-signed EK certificate is sufficient to form strong trust in keys or key policies (although the TPM itself can also be physically compromised).

Enterprise Asset Management with EK Certificates



Enterprise administrators can export the EK_{PUB} from all machines that are recognized corporate assets. The list of EK_{PUB} is used on the enterprise CA to issue enterprise EK certificates. To shortcut this process, the enterprise could request from the OEM the list of all EK_{PUB} of any batch of machines they order before it arrives on the loading dock.

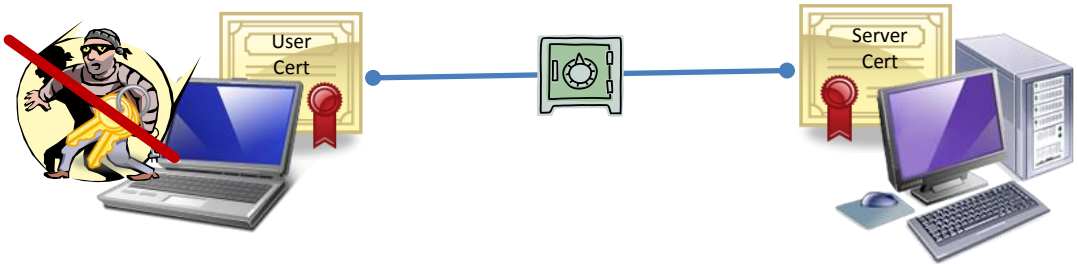
The CA will issue TCG-compliant EK certificates and publish them. When the machine is connected to the network, an automated utility will read the EK_{PUB} from the Platform Crypto Provider, look up the cert in the CA cert database, and import it into the EK cert store. Because the EK of a platform never changes, this would also work if a machine was wiped and reimaged and the TPM was cleared.

The EK certificate lifetime could be set to the intended lifetime of the asset so that decommissioned machines eventually would lose enterprise trust. The enterprise administrator may renew the EK certificate for particular machines that are used beyond the decommissioning date.

It is conceivable that even employees would be allowed to request EK certificates for devices—for example, to enroll their home machines or personal devices to be trusted by the enterprise. Because the EK certificate request would be issued in their name, the administrator can later revoke the certificates for all personal devices if an employee leaves the company.

EK certificates are the static trust anchor on which all of the following scenarios are based.

Retirement of User Names and Passwords for Web Authentication with Mutual SSL

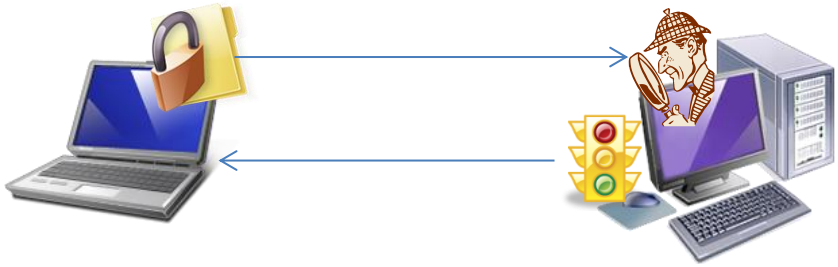


Users can be careless about creating and memorizing strong passwords for web services. Therefore, instead of having the user create a user name and password, the website could use a control that uses the web certificate enrollment feature in Windows to issue a user certificate to the machine. This would happen entirely without interaction with the user, unless the user demands a PIN on the key. The CA server of the website would issue a user certificate and place that in the user's local certificate store.

When the user visits the website again, the server will demand SSL mutual authentication and the web browser will locate the user certificate from the cert store. The browser will access the private key in the Platform Crypto Provider and answer the challenge, optionally obtaining the PIN from the user. The PIN that a user may have associated with the certificate can be a low entropy value, because it is protected from dictionary attacks. Also, the optional PIN is only consumed locally; there is no exposure to the network.

If a client is under attack, the malware may gain access to the user's keys and potentially use them, but will not be able to export the credentials off the machine. This forces malware to remain on the machine and run code whenever it attempts to use the user's credentials. This creates ample opportunity for detection of the malware. When the malware eventually is removed, the user can be assured that his credentials remained safe throughout the attack.

Remote Platform Attestation for Malware Detection



An antivirus (AV) software vendor can provide a service for remote rootkit and bootkit detection. The user can download client software that will create an AIK on the Platform Crypto Provider with the AV server. This procedure may or may not use an EK certificate, if the platform has one. Because the user has the greatest interest in ensuring that the platform attestation is integrity-protected, the user will ensure that the EK certificate is legitimate.

The AV vendor will be able to identify the machine persistently by the EK_{PUB} and can validate whether a service contract is in place. This binding will persist across OS reinstallation and the user will not have to memorize any account details. The AV vendor can also monitor with the EK_{PUB} how many machines the user has enrolled in the service.

The AV client software will enable integrity measurements in the Boot Configuration Database (BCD) of the machine, generate an AIK on the Platform Crypto Provider for the user, and activate it with the AV server. The AV server may issue an AIK certificate or store the public key in a database with the user record as trusted AIK for this machine. The AV client software may register the AIK so the TPM driver will generate trust points on every boot and resume, or else sets a trigger, to produce its own quote at these times. The system will produce the integrity measurements with a trust point if the AIK was registered after the next reboot.

There are two different possible modes of attestation: an incremental online model and an on-demand "big bang" model.

Incremental Online Model

In the incremental online model, the AV client software does not register the AIK to generate trust points. The user or the locally installed AV software has to trigger the AV client software to contact the server for a nonce every time the system boots or resumes from hibernation. The client software will generate a quote with the nonce and send the attestation blob and AIK certificate or public key to the AV server for inspection. The server will recognize the AIK certificate or look up the AIK_{PUB} in the database and validate the integrity of the attestation data. Then it will inspect the log and sign the TPM Event counter value to indicate that the log was validated, and send that value back to the client for storage. The last signed counter value is sent along with the attestation blob on every hibernation resume, so the server can recognize that the previous state had been validated.

The advantage of this model is that the client software usually has to send only one log at a time, but has to do so every time the software is triggered. If the software is triggered when the machine is offline, the AV client software may reuse the last valid nonce for the quote and queue the log until it is online again, to maintain the Root of Trust.

On-Demand Big Bang Model

In this model, the AV registers the AIK and has the TPM driver generate the trust points on its behalf. The AV client software only runs on demand when the user requests an inspection of the machine configuration. The AV client software will retrieve the initial boot log and all resume logs between it and the current state from the log archive. It will send all logs and the AIK certificate or AIKPUB to the AV service for inspection. The server will validate the integrity of all provided logs as above, inspect all entries, and verify that the logs are contiguous from the last full boot (a "big bang" effort). If this validation is successful, the server will issue a nonce to the client that will then perform a quote. Because the server cannot determine whether the machine's current state matches the last log provided, the client will generate an attestation structure with the quote and send that back to the server. The server will validate integrity and nonce in the provided attestation and then simply compare the PCRs in the quote with the PCRs from the last log. If they match, the client has shown that it is in the state of the last validated log and its configuration is valid.

Returning Results to the User

The server can use any means to communicate back to the client about the findings in the evaluation. Out-of-band communication is preferred for this, because malware that has hijacked the machine could show the user anything the user expects to see. The server, for example, could provide a response as an SMS text message or as an email.

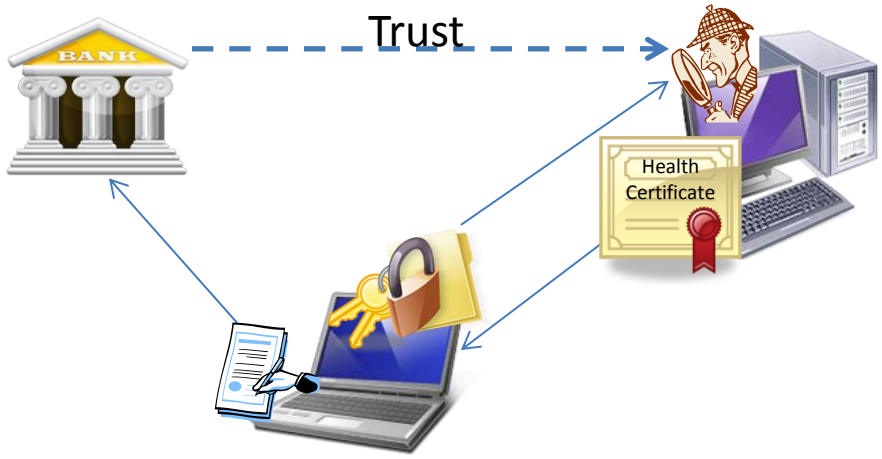
Platform Health Certificates

This scenario addresses the issue described in the previous section: The server does not have a reliable way to communicate platform validation results back to the user, because malware could have hijacked the user interface.

This can be addressed if the server issues to the client a certificate of health that the client needs for performing further operations on the network (such as cosigning bank transactions, for example). The bank requires user-signed transactions that originate from a secure machine. The bank has recognized a secure signing key from the user on a smart card, for example, but has no idea what a secure system is or how to validate that. However, the AV service *does* know and the bank may trust a set of AV services.

There are two slightly different approaches to how a health certificate may be issued: a trust model and a hostage model.

Trust Model

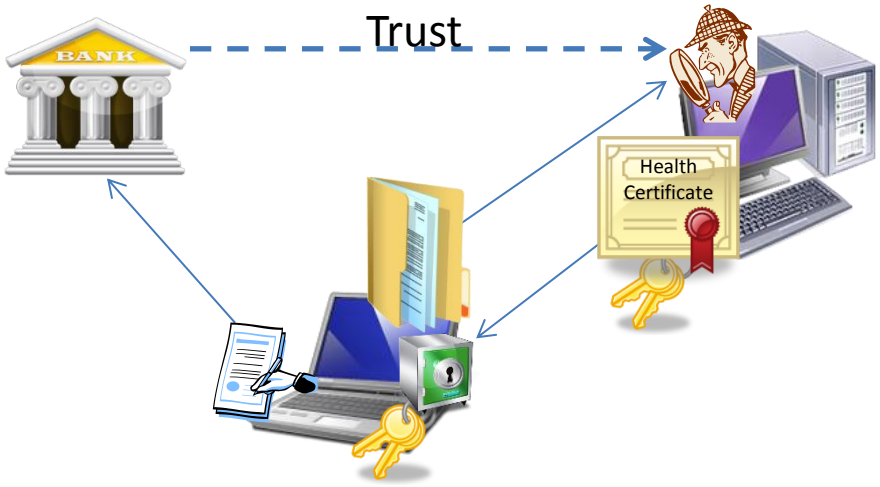


The trust model is based on the previous scenario where the server has to form full trust in the client's keys and measurements. This scenario will work very well for an established AV service. As an additional step, the AV client software will generate a nonexportable key on the Platform Crypto Provider that is bound to the current measurements and create a key certification with the AIK, using the same nonce that the server provided for the platform attestation. This key will remain usable only as long as the machine remains in the same state as it was during the attestation phase.

The AV client software will send the key attestation to the server for validation and the server will check the integrity of the key attestation and the nonce in it, check that the key is not exportable, and validate that the PCRs from the last log match the PCR policy on the key. When the AV server confirms these facts, it will issue a health certificate for the key that is signed with the certificate of the AV service and communicate the certificate to the client.

The client will add the cert to the Windows cert store and associate it with the key handle in the Platform Crypto Provider. Every time the machine reboots, the client software has to create a new key, generate a key attestation, dispose of the previous health certificate, and add the new one.

Hostage Model



In the hostage model, the client registers the SRK_{PUB} with the AV service at the time of provisioning, using key attestation. The AV service then generates a software key and creates a health certificate for it. The health certificate is sent to the client and may be installed in the Windows cert store.

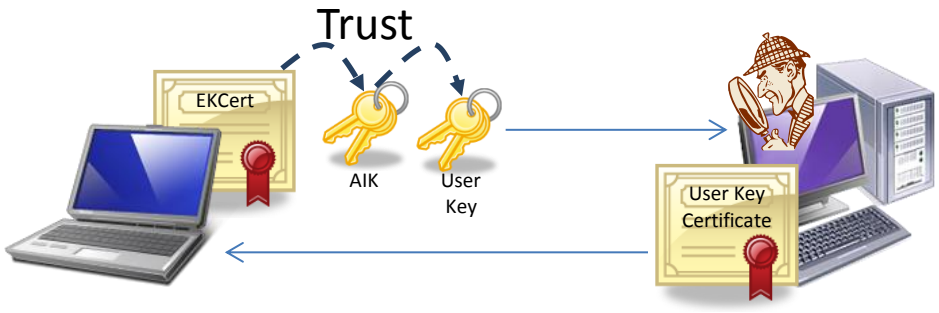
The AV client software then will collect the WBCL from the platform and send the log to the server. No trust point in the log or attestation is required. The server will inspect the log and calculate the PCRs from it. The server at this point cannot determine the state in which the client claims to be. The server then wraps the software key to a PCR-bound platform key with the SRK_{PUB} that is associated with the machine and sends the key unprotected to the AV client software. This key is the *hostage* that the client wants access to. The client gets to use the private key only if the claimed log is backed up by the PCRs in the TPM. If the client provided incorrect information about its configuration, it received a valid key that it will never be able to use.

The client software will import the key into the Platform Crypto Provider and associate it with the health certificate in the certificate store. The server will continue to produce PCR-bound keys for the client every time it is rebooted or resumed from hibernation. Because the key material of the wrapped platform key never changes, the certificate in the Windows certificate store does not have to be replaced. As a result, expensive key creation is avoided and attestation time is saved, replaced with one encryption operation on the main processor of the AV service.

Using the Health Certificate with a Third Party

Because the key associated with the health certificate is bound to the exact platform configuration, the user may use the health certificate to cosign (for example) a banking transaction that the user already signed with a bank smart card. The transaction and the health certificate are sent to the third-party service. The third-party will be able to verify the user signature with the user certificate and the platform signature with the health certificate. The Root of Trust of the health certificate links back to the AV service that is also trusted by the third-party service, giving it the reassurance that the machine is in a known good state, without understanding what a known good state for this machine is.

Certificate Enrollment with Key Origination Proof



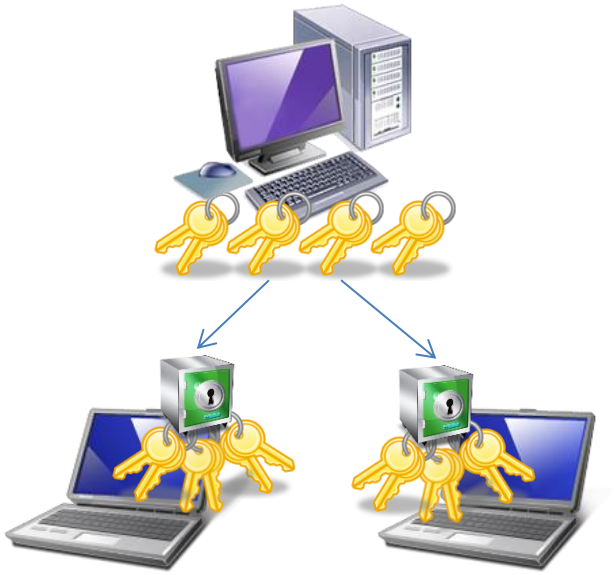
An enterprise CA may have certain policy requirements for particular certificates that enable the client to do remote network access, IPSEC, or user authentication. Currently, certificates like this are mainly issued to a physically present smart card to be sure that the key actually is created on a device that provides strong binding of the key material.

A nonexportable key on the Platform Crypto Provider satisfies such a policy. Assuming that the machine has a TPM manufacturer-issued EK certificate or better an enterprise issued EK certificate as described above, the enterprise could enroll an AIK on the platform. This AIK then may be used by the enrollment agent on the machine to create additional evidence about the origin of the key material and key properties. To allow provisioning of these certificates, the enrollment agent that creates the certificate request would attach a noncritical extension that provides the key attestation data for the key that is held in the Platform Crypto Provider. In addition to the key attestation data, the AIK certificate or its digest is added to the certificate request.

When the enrollment client has submitted the certificate request to the CA, an optional plug-in in the CA could handle the noncritical extension with the attestation data in the certificate request, validate the integrity of the attestation data, and then validate that key properties satisfy the policy requirements in the certificate template. If the validation is successful, the plug-in would permit the issuance of a certificate.

The provisioning of the certificate would continue in the same way that certificate enrollment was done before. Origination proof makes sense only for nonexportable TPM-generated keys, which are likely used for signature operations. Encryption keys, which are usually imported or created with CA escrow, have duplicates that exist outside the TPM and an origination proof has little meaning.

Secure Key Roaming



This scenario addresses a growing need for users who use multiple machines and want access to all keys on all machines, while still being strongly protected against malware attacking the key material. The user is enrolled in a key roaming service that is hosted in a secure location and has access to all of the user's key material and certificates.

When the user receives a new machine from the enterprise administrator, the user authenticates to the key roaming server and attests the SRK_{PUB} to it, creating an AIK by using the enterprise EK certificate. Once the new machine has been identified as a corporate-approved asset, the server can wrap all of the user's key blobs with the SRK and package them with the corresponding certificates. Depending on a per key policy, the server may wrap particular keys with the user's PIN.

The client-side software will import the key blobs in the Platform Crypto Provider and associate the keys with the certificates that are imported into the user's certificate store. At this point, the user has access to all certificates on the machine, while the keys have been fully protected in transit and on the machine. Even if the machine was malware-infested at this time, malware would not have been able to access the user's key material to export it. When the user starts to use a second device, the user would simply go through the same steps to get the keys provisioned to that enterprise-approved machine. It is important to note that the key roaming service remains in control of where the key material may go. If the user attempts to request keys for a machine that is not enterprise-approved, the server would detect that when inspecting the EK certificate and deny the user's request.

An interesting detail is that the keys are persistently bound to the TPM on the local machine, allowing continued key access even if the user has disconnected from the network. The caveat is that when the user changes the PIN for a key on the server, the server has to rewrap all keys generated for each platform and publish them on the machines for the clients to pick up.

Another important factor is PIN reset on keys. If the user has lost the PIN for an important encryption key that cannot be used without one, the user can authenticate back to the roaming service, change the PIN,

and request a new wrapped key to be exported to the user's machine, giving the user access to the key again. It is noteworthy that the PIN on a TPM key can never be *changed* because the old key blob with the previous PIN remains valid and loadable. This means changing a PIN is actually an action to add a PIN.

Especially interesting is an online model where the keys are not statically wrapped for the client as described above, but dynamically wrapped with the client's current PCRs. The server may or may not inspect the WBCL to determine whether the machine is trusted, but it would bind all key blobs to the current configuration the client claims to be in. This would produce keys that are usable only until the machine reboots or goes into hibernation. When the machine shuts down or goes into hibernation, all keys are invalidated. When the machine reboots or resumes from hibernation, the new log would be provided to the roaming service and it would wrap all keys for the new configuration. This would also work as an automatic key revocation[2] mechanism. If the PIN on a key was changed in this case, the key with the old PIN would function only until the machine hibernates or is rebooted and then it would stop functioning.

[2] Not to be mistaken with certificate revocation.