

TPM Base Services

Article • 08/19/2020

Purpose

The Trusted Platform Module (TPM) Base Services (TBS) feature centralizes TPM access across applications.

The TBS feature runs as a system service in Windows Server 2008, Windows Vista, or newer operating systems. It provides services as an API exposed through remote procedure calls (RPC). The TBS feature uses priorities specified by calling applications to cooperatively schedule TPM access.

ⓘ Note

The TPM can be used for key storage operations. However, developers are encouraged to use the **Key Storage APIs** for these scenarios instead. The Key Storage APIs provide the functionality to create, sign or encrypt with, and persist cryptographic keys, and they are higher-level and easier to use than the TBS for these targeted scenarios.

Developer audience

TBS is intended for use by developers of applications based on the Windows operating systems. Developers should be familiar with the C and C++ programming languages and the Microsoft Windows programming environment.

Run-time requirements

The TBS feature requires at least Windows Server 2008 or Windows Vista operating system. For information about run-time requirements for a particular programming element, see the Requirements section of the reference page for that element.

In this section

Topic	Description
About TBS	Key concepts and a high-level view of the TBS feature.

Topic	Description
Using TBS	TBS processes and procedures for using the TBS API.
TBS Reference	Documentation about the TBS functions, structures, and return codes.

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

About TBS

Article • 02/20/2020

The TPM Base Services (TBS) feature is a system service that allows transparent sharing of the Trusted Platform Module (TPM) resources. It simultaneously shares the TPM resources among multiple applications on the same physical machine, even if those applications run on different virtual machines. Starting with Windows 8 and Windows Server 2012, TBS comes pre-installed on all systems with a TPM.

The Trusted Computing Group (TCG) defines a Trusted Platform Module that provides cryptographic functions designed to provide trust in the platform. Because the TPM is implemented in hardware, it has finite resources. The TCG also defines a software stack that makes use of these resources to provide trusted operations for application software. However, no provision is made for running a TSS implementation side-by-side with operating system software that may also be using TPM resources. The TBS feature solves this problem by enabling each software stack that communicates with TBS to use TPM resources checking for any other software stacks that may be running on the machine.

The TPM specification and TCG Software Stack (TSS) specification are available at <https://www.trustedcomputinggroup.org>.

TBS is implemented as an out-of-process service that accepts commands that use an RPC service. A dynamically linked library presents the C language interface and communicates with the TBS service.

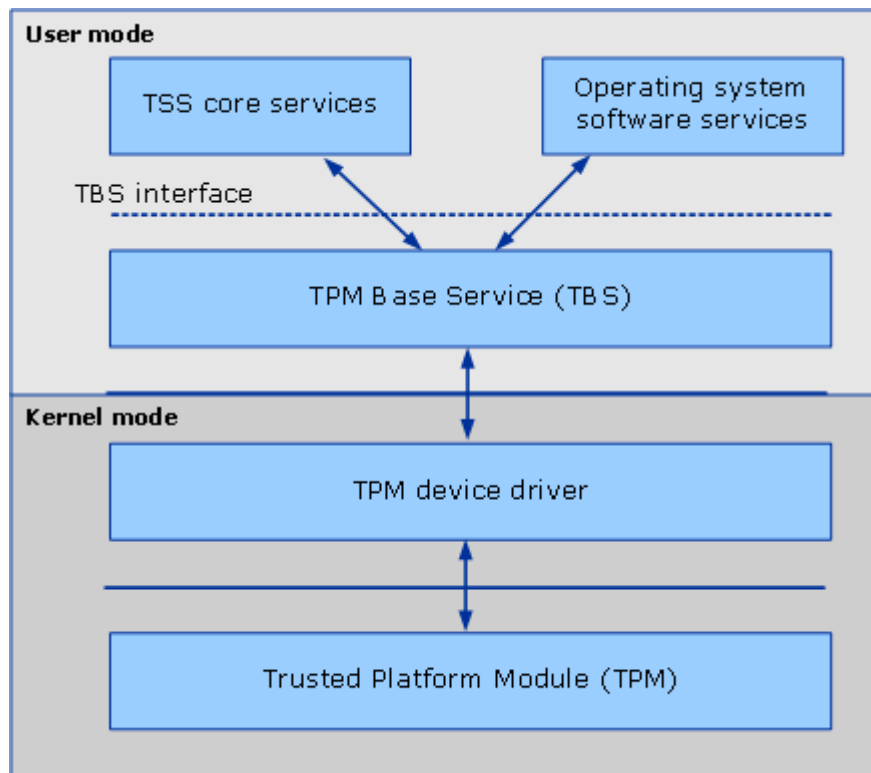
ⓘ Note

The TBS service only accepts RPC requests from the local machine.

The primary goals of the TBS are to:

- Provide efficient sharing of limited TPM resources, such as key slots, authorization sessions slots, and transport slots.
- Provide prioritized and synchronized access to TPM resources between multiple instances of TPM software stacks.
- Provide appropriate management of TPM resources across power states.
- Prevent TPM software stacks from accessing TPM commands that should be restricted, either because of platform limitations or administrative requirements.

The following illustration shows the relationship of the TBS to the TPM.



Feedback

Was this page helpful?

☐ Yes

☐ No

[Get help at Microsoft Q&A](#)

Using TBS

Article • 08/23/2019

The TPM Base Services feature is divided into four functional areas:

- [Resource Virtualization](#)
- [Command Scheduling](#)
- [Power Management](#)
- [Command Blocking](#)

To ensure that different entities cannot access each other's resources, each command submitted to the TBS is associated with a specific entity. This is accomplished by creating one or more contexts for an entity, which are then associated with each subsequent command submitted by that entity. Each command includes a context object, which allows the TBS to execute TPM commands under the appropriate context. All objects created by commands are flushed from the TPM when the context is closed.

An entity creates the context before it first accesses the TBS and maintains the context until it is finished performing TBS accesses. For example, in the case of a TSS, the TCG core services (TCS) feature of the TSS would create a TBS context when it starts up, and it would keep that context active until it shuts down.

For Windows Server 2008 and Windows Vista, the TBS restricts access to the TBS API to the Administrators, NT AUTHORITY\LocalService, and NT AUTHORITY\NetworkService accounts. By default, these accounts are the only ones that can connect to TBS and create contexts. Access restrictions can be modified by creating a registry key **Access** with a string (REG_SZ) registry value name **SecurityDescriptor**

Data type

REG_SZ

under it as follows:

```
HKEY_LOCAL_MACHINE
  Software
    Microsoft
      TPM
        Access
          SecurityDescriptor = SecurityDescriptor
```

Example:

O:BAG:BAD:(A;;0x00000001;;;BA)(A;;0x00000001;;;NS)(A;;0x00000001;;;LS)

By default, the maximum number of contexts supported by the TBS is 25. This number can be altered by creating or modifying a **DWORD** registry value named **MaxContexts** under **HKEY_LOCAL_MACHINE\Software\Microsoft\Tpm**. Real-time TBS context usage can be observed by using the performance monitor tool to track the number of TBS contexts.

For Windows 8, Windows Server 2012 and later, the TBS allows access to standard users and administrators. The **SecurityDescriptor** and **MaxContexts** registry keys became obsolete. For Windows 8, Windows Server 2012 and later, the TBS restricts access to certain commands using Command Blocking.

For Windows 10, version 1607, the TBS allows access from AppContainer applications. For each TPM version, the keys **BlockedAppContainerCommands** and **AllowedW8AppContainerCommands** have been added with the matching lists of blocked and allowed TPM commands, respectively.

For Windows 10, version 1803, the registry keys under **AllowedW8AppContainerCommands** are no longer supported.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Resource Virtualization

Article • 08/23/2019

The primary function of the TBS is to efficiently share certain limited TPM resources: keys, authorization, and transport sessions.

When an instance of one of these resources is created, the TBS creates a virtual instance of the resource, and returns a handle to this virtual instance in the result command stream (rather than the actual handle instance returned by the TPM). The TBS maintains a mapping between the virtual handle and the actual handle internally. If the TPM runs out of space to hold more resources of a given type, existing instances of the resource are selectively saved and evicted until the TPM can hold the new resource. When the old resources are required again, the TBS loads the saved contexts (saving and evicting other resources if necessary) before submitting the command.

All virtualization in the TBS is performed on behalf of a specific context. Each context is only allowed to access virtual resources created specifically on its behalf, as well as the physical resources on the TPM that corresponds to those virtual resources. By default, the total number of all virtualized resources is limited to 500. This number can be altered by creating or modifying a **DWORD** registry value named **MaxResources** under **HKEY_LOCAL_MACHINE\Software\Microsoft\Tpm**. Real-time TBS resource usage can be observed by using the Performance Monitor tool to track the number of TBS resources. This restriction became obsolete with Windows 8 and Windows Server 2012.

Limited TPM resources that are not virtualized by the TBS (such as counters and NV store) must be cooperatively shared among software stacks.

ⓘ Note

This handle virtualization causes commands that include key handles in the calculation of HMAC authorization parameters to fail. As a result, many commands deprecated in TPM version 1.2 cannot be used by application software in the TBS environment.

Resource Limits

The TPM enables callers to query its capabilities to determine how much space is available for certain types of resources. Some of these resource limits, such as the

amount of space available for keys, authorization sessions, and transport sessions, are effectively extended by the TBS through virtualization. TBS limitations, which are controlled by the **MaxResources** registry setting, are usually far larger than the actual limitations of the underlying TPM hardware. No mechanism is supplied for querying TBS limitations separately from the TPM hardware limits. This TBS limitation became obsolete with Windows 8 and Windows Server 2012.

Keys

The TBS virtualizes key handles so that keys can be transparently unloaded from the TPM when they are not being used and loaded back onto the TPM when they are needed. When a key is created, the TBS associates a virtual handle with the loaded key. The same virtual handle is used for the lifetime of the resource. Virtual key handles are only valid in the context that created them, and the associated resources are not preserved beyond the life of the context.

- Creating Keys with TPM_LoadKey2

If a key is created by using the TPM_LoadKey2, TPM2_CreatePrimary, TPM2_Load, or TPM2_LoadExternal command, the TBS replaces the handle in the return byte stream with a virtual key handle of its choosing. As a result, the TBS ensures that each virtual handle is unique. If the TBS detects a collision (an extremely unlikely event), the TBS unloads the key from the TPM and informs the calling software. The software can then resubmit the operation. This process can be repeated until the TBS gets a unique key handle.

- Clearing Keys

The TBS invalidates the virtual key handle when it receives a TPM_FlushSpecific or TPM2_FlushContext message for that virtual handle from the client context. If the key is physically present on the TPM when the flush message is received, the TBS flushes the key from the TPM at that time.

- Temporarily Removing Keys

When evicting a key from the TPM to make room for a new item, the TBS executes a TPM_SaveContext or TPM2_ContextSave command on the key before evicting it.

- Restoring Keys

When a command that references a loaded key is submitted to the TBS, it ensures that the key is physically present on the TPM. If the key is not present, the TBS restores it with a call to TPM_LoadContext or TPM2_ContextLoad. If a key cannot

be restored to the TPM, the TBS returns `TPM_E_INVALID_KEYHANDLE` as the TPM result.

The TBS replaces each virtual handle associated with a key in a command stream with the physical handle of the key loaded on the TPM. If a command is submitted with a virtual handle that is not recognized by the TBS in the context of the caller, the TBS formats an error stream for the caller with `TPM_E_INVALID_KEYHANDLE`.

Authorization Sessions

Authorization sessions are created by calling `TPM_OIAP`, `TPM_OSAp`, or `TPM_DSAP`. In each case, the return byte stream contains the physical TPM handle of the newly created authorization session. The TBS replaces this with a virtual handle. When the authorization session is subsequently referenced, the TBS replaces the virtual handle in the command stream with the physical handle of the authorization session. The TBS ensures that the lifetime of the virtual authorization session matches that of the physical authorization session. If a client attempts to use an expired virtual handle, the TBS formats an error stream with error `TPM_INVALIDAUTHHANDLE`.

Session slots are limited, and the TBS may run out of external slots in which to save authorization session contexts. If this happens, the TBS chooses an authorization session to invalidate so that the new context can be successfully saved. An application that attempts to use the old context will need to re-create the authorization session.

The TBS invalidates the virtual authorization session when any of the following cases occur:

- The continue-use flag associated with the authorization session in the returned command stream from the TPM is **FALSE**.
- A command that uses an authorization session fails.
- A command is executed that invalidates the authorization session associated with the command (such as `TPM_CreateWrapKey`).
- A key associated with an `OSAP` or `DSAP` session is evicted from the TPM with a call to `TPM_FlushSpecific` or `TPM2_FlushContext` (without regard to whether this command originated with the TBS or with higher-level software).

The TBS will automatically resynchronize the authorization sessions after successful execution of certain nondeterministic commands to ensure that the TBS state remains consistent with the TPM state. The affected commands are:

- `TPM_ORD_Delegate_Manage`

- TPM_ORD_Delegate_CreateOwnerDelegation
- TPM_ORD_Delegate_LoadOwnerDelegation

In each of the following cases the authorization session on the TPM is flushed automatically by the TPM:

- Creating Authorization Sessions

Virtual authorization session handles are only valid in the context that created them, and the associated resources are not preserved beyond the life of the associated context.

- Clearing Authorization Sessions

The TBS invalidates the virtual authorization session if it receives a TPM_FlushSpecific or TPM2_FlushContext command for the virtual handle from the client context. If the authorization session is physically present on the TPM when the flush command is received, the TBS flushes the physical session from the TPM immediately.

- Temporarily Removing Authorization Sessions

When evicting an authorization session from the TPM to make room for a new entity, the TBS executes TPM_SaveContext or TPM2_ContextSave on that authorization session.

- Restoring Authorization Sessions

When an authorized TPM command is submitted to the TBS, the TBS ensures that all virtual authorization sessions referred to in the command are physically present on the TPM. If any of the authorization sessions are not present, the TBS restores them with a call to TPM_LoadContext or TPM2_ContextLoad. If an authorization session cannot be restored to the TPM, the TBS returns TPM_E_INVALID_HANDLE as the TPM result.

The TBS replaces each virtual handle associated with an authorization session in a command stream with the physical handle of the authorization session loaded on the TPM.

If a command is submitted with a virtual handle that is not recognized by the TBS in the context of the caller, the TBS formats an error stream for the caller with the error TPM_E_INVALID_HANDLE.

Transport Sessions

ⓘ Note

The handling of transport sessions as described here is specific to Windows Vista and Windows Server 2008.

Transport sessions are a mechanism provided by the TPM that enables a software stack to encrypt data in a command as it passes between the software and the TPM. This prevents an adversary from intercepting the data as it passes over the hardware bus.

ⓘ Important

Only the payload data is encrypted. The commands being executed can still be identified.

Unfortunately, this mechanism also prevents the TBS from examining payload data. In most cases this is not an issue because the TBS only modifies handles, not payload data. However, in the case of TPM_LoadContext for instance, the returned resource handle is covered by the encryption. Therefore, the TBS prevents attempts to perform a TPM_LoadContext operation covered by a transport session.

The TBS blocks the following commands under transport session:

- TPM_EstablishTransport
- TPM_ExecuteTransport
- TPM_Terminate_Handle
- TPM_LoadKey
- TPM_EvictKey
- TPM_SaveKeyContext
- TPM_LoadKeyContext
- TPM_SaveAuthContext
- TPM_LoadAuthContext
- TPM_SaveContext
- TPM_LoadContext
- TPM_FlushSpecific

When any one of these commands is covered by a transport session, the TBS returns TPM_E_EMBEDDED_COMMAND_UNSUPPORTED as the TPM result.

Transport session handles are virtualized in a manner similar to key handles and authorization handles. There are a limited number of saved transport session context slots available on the TPM.

The TBS invalidates the virtual transport session if any of the following cases occurs:

- The continue-use flag associated with the transport session in the return command stream from the TPM is **FALSE**.

As with authorization sessions above, the TBS will automatically resynchronize transport sessions after successful execution of certain nondeterministic commands to ensure that the TBS state remains consistent with the TPM state. The affected commands are:

- TPM_ORD_Delegate_Manage
- TPM_ORD_Delegate_CreateOwnerDelegation
- TPM_ORD_Delegate_LoadOwnerDelegation

In each of these cases, the transport session on the TPM will be flushed automatically by the TPM:

- Creating Transport Sessions

The TBS creates a virtual handle for each transport session created by a client. Virtual transport handles are only valid in the context that created them, and the associated resources are not preserved beyond the life of the associated context.

- Clearing Transport Sessions

The TBS invalidates the virtual transport session if it receives a TPM_FlushSpecific command for the virtual handle from the client context. If the transport session is physically present on the TPM when the flush command is received, the TBS flushes the physical session from the TPM immediately.

- Temporarily Removing Transport Sessions

When evicting a transport session from the TPM to make room for a new entity, the TBS executes TPM_SaveContext on that transport session.

- Restoring Transport Sessions

When a TPM_ExecuteTransport command is submitted to the TBS, the TBS ensures that the transport session referred to in the command is physically present on the TPM. If the transport session is not present, the TBS restores it with a call to TPM_LoadContext.

The TBS replaces the virtual handle associated with the transport session in a command stream with the physical handle of the transport session loaded on the TPM. If a command is submitted with a virtual handle that is not recognized by the TBS in the context of the caller, the TBS formats an error stream for the caller by using TPM_E_INVALID_HANDLE.

Exclusive Transport Sessions

Exclusive wrapped transport sessions are a way for top-level software to determine whether any other software has accessed the TPM during a chain of commands. They do not prevent other software from accessing the TPM, they just give the creator of the transport session a means of determining that some other access occurred. The TBS does not do anything specific to hinder exclusive transport sessions, but it is somewhat incompatible with them. The TBS attempts to duplicate the natural behavior of the TPM by being transparent, so it does not favor commands from contexts that establish an exclusive transport session. For example, if client B submits a command when client A is in an exclusive transport session, then it will invalidate the exclusive transport session of client A.

Commands initiated by TBS can also terminate the exclusive transport session. This happens when client A is in an exclusive transport session, and a command executed by client A calls for a resource that is not physically present in the TPM. This situation triggers the TBS virtualization manager to execute a TPM_LoadContext command to supply that resource, which terminates the exclusive transport session of client A.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Command Scheduling

Article • 08/23/2019

Command Scheduling

Commands can be submitted to the TBS from multiple contexts at any time. However, a physical TPM can only process one command at a time, and some commands can run for a significant period of time. When multiple commands are pending, the TBS decides which command to send to the TPM. When a command is submitted, each command has a priority associated with it. The TBS uses these priorities to determine which pending command to submit whenever the TPM is free. The four levels of priority are low, normal, high, and system. Applications should use low, normal, or high priority. Although high-priority commands are usually executed before low-priority commands, the TBS command scheduler has an aging policy that eventually gives very high priority to commands that have been blocked for significant periods of time.

Context Management

When a TPM_SaveContext or TPM2_ContextSave command is submitted with a virtual handle to a resource that the TBS is managing, the TBS executes the appropriate command on the physical resource and returns the result to the caller if the resource is physically present in the TPM. If the resource is not physically present in the TPM, the preprocessing of the TPM_SaveContext or TPM2_ContextSave command causes the resource to be reloaded, at which point the context will be saved and returned to the caller. If the resource being saved is of a type that is normally flushed automatically from the TPM after saving, the TBS invalidates the virtual resource upon completion of this command. When a TPM_LoadContext or TPM2_ContextLoad command is submitted to the TBS, the TBS executes the command immediately. If the command completes successfully, the TBS virtualizes the resource handle and passes the resulting TPM byte stream to the caller. When a TPM_FlushSpecific or TPM2_FlushContext command is submitted with a virtual handle to a resource that the TBS is managing, the TBS executes a TPM_FlushSpecific or TPM2_FlushContext command to evict the resource from the TPM if it is physically present. In this case, the TBS invalidates the virtual resource and returns the result of the flush command to the caller. If the resource is not physically present in the TPM, the preprocessing of the TPM_FlushSpecific or TPM2_FlushContext command will load the resource that corresponds to the virtual handle, but it will then be flushed when the command is actually processed. The TBS does not support the TPM_KeyControlOwner bit or the keepHandle functionality of the TPM_LoadContext

operation, so it will not give a resource the same handle it had before the context was saved.

Other Commands

Commands that are not mentioned in this documentation are processed by replacing virtual key handles with physical key handles (if necessary) and submitting them to the TPM. This is done after performing the appropriate operations to ensure that resources being used are physically present on the TPM. No additional special processing is performed. The TBS does not attempt to improve the fidelity of the virtualization by modifying data passed back from the TPM as part of a TPM_GetCapabilities or TPM2_GetCapability query. The TBS also supports TCG Physical Presence commands. The TBS acts as a pass-through for Physical Presence commands; no special processing is performed by the TBS itself.

Cleanup

When a context exits, all physical and virtual resources associated with it are cleaned up so they no longer consume system resources.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Power Management (TPM Base Services)

Article • 12/11/2019

The TBS receives power management events. When an indication is received that the TPM or other parts of the platform are about to enter a power state in which execution will be interrupted or TPM state will be lost, the TBS checks to determine whether the currently executing command is likely to finish before the system powers down. In general, the TBS allows short and medium duration commands to finish, but cancels long duration commands. After the command has returned, the TBS stops sending new commands to the TPM and prepares itself for hibernation. When power is restored, the TBS returns the result of the command to the caller, and then proceeds with processing pending TBS commands. The TBS power management code runs asynchronously, so it can handle power management requests even if the TPM is processing a long command.

When a computer enters sleep states, including S3 (sleep) and S4 (hibernation), the TPM is powered off. Thus, all nonpersistent TPM states are lost. Before entering these states, application software is expected to prepare for the loss of volatile TPM states. When the system returns from a sleep state, the TBS synchronizes with the TPM so that the TBS state is consistent with the TPM state. Application software may need to reissue commands that were interrupted.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

Command Blocking

Article • 06/17/2020

To preserve integrity of operations, certain TPM commands are not allowed to be executed by software on the platform. For example, some commands are only executed by system software. When the TBS blocks a command, an error is returned as appropriate. By default, the TBS blocks commands that could impact system privacy, security, and stability. The TBS also assumes that other parts of the software stack may restrict access to certain commands to authorized entities.

For TPM version 1.2 commands, there are three lists of blocked commands: a list controlled by group policy, a list controlled by local administrators, and a default list. A TPM command is blocked if it is on any of the lists. However, there are group policy flags to allow the TBS to ignore the local list and the default list. The group policy flags can be edited directly or accessed through the Group Policy Object Editor.

ⓘ Note

The list of locally blocked commands is not preserved after an upgrade to the operating system. Commands that are blocked on the Group Policy list are preserved.

For TPM version 2.0 commands, the logic for blocking is inverted; it uses a list of allowed commands. This logic will automatically block commands that were not known when the list was first made. When commands are added to the TPM specification after a version of Windows has shipped, these new commands are automatically blocked. Only an update of the registry will add these new commands to the list of allowed commands.

Starting with Windows 10 1809 (Windows Server 2019), allowed TPM 2.0 commands can no longer be manipulated through registry settings. For these Windows 10 versions, the allowed TPM 2.0 commands are fixed in the TPM driver. TPM 1.2 commands can still be blocked and unblocked through registry changes.

Direct Registry Access

The Group Policy flags are under registry key

`HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Tpm\BlockedCommands`.

To determine which lists should be used to block TPM commands, there are two **DWORD** values that are used as Boolean flags:

- "IgnoreDefaultList"

If set (value exists and is nonzero), the TBS ignores the default blocked-commands list.

- "IgnoreLocalList"

If set (value exists and is nonzero), the TBS ignores the local blocked-commands list.

Group Policy Object Editor

To access the Group Policy object editor

1. Click **Start**.
2. Click **Run**.
3. In the **Open** box, type **gpedit.msc**. Click **OK**. The Group Policy object editor opens.
4. Expand **Computer Configuration**.
5. Expand **Administrative Templates**.
6. Expand **System**.
7. Expand **Trusted Platform Module Services**.

The lists of specific blocked TPM1.2 commands can be edited directly in the following locations.

- Group policy list:

```
HKEY_LOCAL_MACHINE
  Software
    Policies
      Microsoft
        Tpm
          BlockedCommands
            List
```

- Local list:

```
HKEY_LOCAL_MACHINE
  SYSTEM
```

```
CurrentControlSet
  Services
    SharedAccess
      Parameters
        Tpm
          BlockedCommands
            List
```

- Default list:

```
HKEY_LOCAL_MACHINE
  Software
    Microsoft
      Tpm
        BlockedCommands
          List
```

Under each of these registry keys, there is a list of registry values of REG_SZ type. Each value represents a blocked TPM command. Each registry key has a "Value name" field and a "Value data" field. Both fields ("Value Name" and "Value data"), should exactly match the decimal value of the TPM command ordinal to be blocked.

The list of specific allowed TPM 2.0 commands can be edited directly in the following location. Under the registry key, there is a list of registry values of REG_DWORD type. Each value represents an allowed TPM 2.0 command. Each registry value has a **name** and a **value** field. The **name** matches the hexadecimal TPM 2.0 command ordinal that should be allowed. The **value** has a value of 1 if the command is allowed. If a command ordinal is not present or has a value of 0, the command will be blocked.

- Default list:

```
HKEY_LOCAL_MACHINE
  Software
    Microsoft
      Tpm
        AllowedW8Commands
          List
```

For Windows 8, Windows Server 2012 and later, the **BlockedCommands** and **AllowedW8Commands** registry keys respectively determine the blocked or allowed TPM commands for administrator accounts. User accounts have a list of blocked or allowed TPM commands in the **BlockedUserCommands** and **AllowedW8UserCommands**

registry keys respectively. In Windows 10, version 1607, new registry keys have been introduced for AppContainer applications: **BlockedAppContainerCommands** and **AllowedW8AppContainerCommands**.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

TBS Reference

Article • 10/24/2019

The Trusted Platform Module (TPM) Base Services (TBS) API supports the following elements.

In this section

Topic	Description
TBS Data Types	TBS defines the following data types.
TBS Functions	TBS provides the following functions.
TBS Return Codes	TBS uses the following codes to indicate the result of a function call.
TBS Structures	TBS supports the following structures.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

TBS Data Types

Article • 12/11/2020

TBS defines the following data types.

Data type	Description
TBS_HCONTEXT	Context object handle.
TBS_COMMAND_PRIORITY	Priority associated with commands.
TBS_COMMAND_LOCALITY	The locality that the command is submitted from.
TBS_OWNERAUTH_TYPE	The type of owner authentication. Windows Server 2008 R2, Windows 7, Windows Server 2008 and Windows Vista: This data type is not available.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	Tbs.h

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

TBS Functions

Article • 08/19/2020

TBS provides the following functions.

In this section

Topic	Description
Tbsi_Context_Create	Creates a context handle that can be used to pass commands to TBS.
Tbsi_Create_Attestation_From_Log	Creates an attestation by extracting a TrustPoint from a TCG log.
Tbsi_GetDeviceInfo	Obtains the version of the TPM on the computer.
Tbsi_Get_OwnerAuth	Retrieves the owner authorization of the TPM if the information is available in the local registry.
Tbsi_Get_TCG_Log	Retrieves the most recent Windows Boot Configuration Log (WBCL), also referred to as a TCG log.
Tbsi_Get_TCG_Log_Ex	Gets the Windows Boot Configuration Log (WBCL), also referred to as the TCG log, of the specified type.
Tbsi_Get_TCG_Logs	Get one or more Windows Boot Configuration Logs (WBCL), also referred to as the TCG logs.
Tbsi_Physical_Presence_Command	Passes a physical presence ACPI command through TBS to the driver.
Tbsi_Revoke_Attestation	Invalidates the PCRs if the ELAM driver detects a policy-violation (a rootkit, for example).
Tbsip_Cancel_Commands	Cancels all outstanding commands for the specified context.
Tbsip_Context_Close	Closes a context handle, which releases resources associated with the context in TBS and closes the binding handle used to communicate with TBS.
Tbsip_Submit_Command	Submits a Trusted Platform Module (TPM) command to TPM Base Services (TBS) for processing.

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

Tbsi_Context_Create function (tbs.h)

Article 10/13/2021

Creates a context handle that can be used to pass commands to TBS.

Syntax

C++

```
TBS_RESULT Tbsi_Context_Create(  
    [in]  PCTBS_CONTEXT_PARAMS pContextParams,  
    [out] PTBS_HCONTEXT        phContext  
);
```

Parameters

[in] pContextParams

A parameter to a [TBS_CONTEXT_PARAMS](#) structure that contains the parameters associated with the context.

[out] phContext

A pointer to a location to store the new context handle.

Return value

If the function succeeds, the function returns TBS_SUCCESS.

If the function fails, it returns a TBS return code that indicates the error.

[Expand table](#)

Return code/value	Description
TBS_SUCCESS 0 (0x0)	The function was successful.
TBS_E_BAD_PARAMETER 2150121474 (0x80284002)	One or more parameter values are not valid.
TBS_E_INTERNAL_ERROR	An internal software error occurred.


2150121473 (0x80284001)	
TBS_E_INVALID_CONTEXT_PARAM 2150121479 (0x80284007)	A context parameter that is not valid was passed when attempting to create a TBS context.
TBS_E_INVALID_OUTPUT_POINTER 2150121475 (0x80284003)	A specified output pointer is not valid.
TBS_E_SERVICE_DISABLED 2150121488 (0x80284010)	The TBS service has been disabled.
TBS_E_SERVICE_NOT_RUNNING 2150121480 (0x80284008)	The TBS service is not running and could not be started.
TBS_E_SERVICE_START_PENDING 2150121483 (0x8028400B)	The TBS service has been started but is not yet running.
TBS_E_TOO_MANY_TBS_CONTEXTS 2150121481 (0x80284009)	A new context could not be created because there are too many open contexts.
TBS_E_TPM_NOT_FOUND 2150121487 (0x8028400F)	A compatible Trusted Platform Module (TPM) Security Device cannot be found on this computer.

Remarks

The [TBS_CONTEXT_PARAMS](#) structure can be provided, with the version field set to TPM_VERSION_12. Applications interacting with version 2.0 TPM 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 a version 1.2 TPM as well (in case the system has no version 2.0 TPM), set the **includeTpm12** field to 1.

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 for both versions and be able to interact with either TPM.

Requirements


 Expand table


Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	tbs.h
Library	Tbs.lib
DLL	Tbs.dll

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

Tbsi_GetDeviceInfo function (tbs.h)

Article02/22/2024

Obtains the version of the TPM on the computer.

Syntax

```
C++

TBS_RESULT Tbsi_GetDeviceInfo(
    [in]  UINT32  Size,
    [out] PVOID   Info
);
```

Parameters

[in] Size

Size of the memory location.

[out] Info

A pointer to a [TPM_DEVICE_INFO](#) structure is returned containing the version information about the TPM. The location must be large enough to hold four 32-bit values.

Return value

If the function succeeds, the function returns TBS_SUCCESS.

If the function fails, it returns a TBS return code that indicates the error.

 Expand table

Return code/value	Description
TBS_SUCCESS 0 (0x0)	The function was successful.
TBS_E_BAD_PARAMETER 2150121474 (0x80284002)	One or more parameter values are not valid.

TBS_E_TPM_NOT_FOUND
2150121487 (0x8028400F)

A compatible Trusted Platform Module (TPM) Security Device cannot be found on this computer.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	tbs.h
Library	Tbs.lib
DLL	Tbs.dll

Feedback

Was this page helpful?

[Yes](#)

[No](#)

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Tbsi_Get_OwnerAuth function (tbs.h)

Article02/22/2024

Retrieves the owner authorization of the TPM if the information is available in the local registry.

Syntax

```
C++

TBS_RESULT Tbsi_Get_OwnerAuth(
    [in]          TBS_HCONTEXT      hContext,
    [in]          TBS_OWNERAUTH_TYPE ownerauthType,
    [out, optional] PBYTE          pOutputBuf,
    [in, out]     PUINT32          pOutputBufLen
);
```

Parameters

[in] hContext

TBS handle obtained from a previous call to the [Tbsi_Context_Create](#) function.

[in] ownerauthType

Unsigned 32-bit integer indicating the type of owner authentication.

[Expand table](#)

Value	Meaning
TBS_OWNERAUTH_TYPE_FULL 1	The owner authorization is full.
TBS_OWNERAUTH_TYPE_ADMIN 2	Note TPM 1.2 only The owner authorization is an administrator.
TBS_OWNERAUTH_TYPE_USER 3	Note TPM 1.2 only The owner authorization is a user.
TBS_OWNERAUTH_TYPE_ENDORSEMENT 4	Note TPM 1.2 only

	The owner authorization is an endorsement authorization.
TBS_OWNERAUTH_TYPE_ENDORSEMENT_20 12	Note TPM 2.0 and later The owner authorization is an endorsement authorization.
TBS_OWNERAUTH_TYPE_STORAGE_20 13	Note TPM 2.0 and later The owner authorization is an administrator.

[out, optional] pOutputBuf

A pointer to a buffer to receive the TPM owner authorization information.

[in, out] pOutputBufLen

An integer that, on input, specifies the size, in bytes, of the output buffer. On successful return, this value is set to the actual size of the TPM ownerAuth, in bytes.

Return value

If the function succeeds, the function returns **TBS_SUCCESS**.

If the function fails, it returns a TBS return code that indicates the error.

 Expand table


Return code/value	Description
TBS_SUCCESS 0 (0x0)	The function was successful.
TBS_E_OWNERAUTH_NOT_FOUND 2150121493 (0x80284015)	The requested TPM ownerAuth value was not found.
TBS_E_BAD_PARAMETER 2150121474 (0x80284002)	The requested TPM ownerAuth value does not match the TPM version.

Remarks

There are additional authorization values, also known as delegation blobs, derived from the full TPM ownerAuth that allow a subset of the TPM functionality to be executed. The administrator can configure the level of ownerAuth that should be locally stored in the registry through Group Policy and the same can be obtained from this API call.



If Active Directory backup of ownerAuth is enabled through Group Policy, the default level of ownerAuth is set as Delegated which means that the full owner auth is removed from the local registry and stored in Active Directory. Only delegation blobs are locally stored in the registry in that case. Although, the level of ownerAuth storage can be explicitly configured to Full resulting in the TPM ownerAuth being locally available in the registry.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	tbs.h
Library	Tbs.lib
DLL	Tbs.dll

Feedback

Was this page helpful?  Yes  No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

Tbsi_Get_TCG_Log function (tbs.h)

Article 03/15/2023

Retrieves the most recent Windows Boot Configuration Log (WBCL), also referred to as a TCG log.

Syntax

C++

```
TBS_RESULT Tbsi_Get_TCG_Log(  
    [in]      TBS_HCONTEXT hContext,  
    [out]     PBYTE        pOutputBuf,  
    [in, out] PUINT32      pOutputBufLen  
);
```

Parameters

[in] hContext

The TBS handle of the context that is retrieving the log. You get this parameter from a previous call to the [Tbsi_Context_Create](#) function.

[out] pOutputBuf

A pointer to a buffer to receive and 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.

[in, out] pOutputBufLen

A pointer to an unsigned long integer 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 *pOutputBuf*. If the function fails, this parameter does not receive a value.

Calling the **Tbsi_Get_TCG_Log** function with a zero length buffer will return the size of the buffer required. **Windows Vista with SP1 and Windows Server 2008:** This functionality is not available.

Return value

Return code/value	Description
TBS_SUCCESS 0 (0x0)	The function succeeded.
TBS_E_INTERNAL_ERROR 2150121473 (0x80284001)	An internal software error occurred. Note If TBS_E_INTERNAL_ERROR is returned, the system event log may contain event ID 16385 from the TBS event source with error code 0x80070032. This may indicate that the hardware platform does not provide a TCG event log to the operating system. Sometimes this can be resolved by installing a BIOS upgrade from the platform manufacturer.
TBS_E_INVALID_OUTPUT_POINTER 2150121475 (0x80284003)	A specified output pointer is not valid.
TBS_E_INVALID_CONTEXT 2150121476 (0x80284004)	The specified context handle does not refer to a valid context.
TBS_E_INSUFFICIENT_BUFFER 2150121477 (0x80284005)	The output buffer is too small.
TBS_E_BUFFER_TOO_LARGE 2150121486 (0x8028400E)	The output buffer is too large.
TBS_E_TPM_NOT_FOUND 2150121487 (0x8028400F)	A compatible Trusted Platform Module (TPM) Security Device cannot be found on this computer.
TBS_E_DEACTIVATED 2150121494 (0x80284016)	The Trusted Platform Module (TPM) Security Device is deactivated. Windows Vista with SP1 and Windows Server 2008: This return value is not available.

Remarks

The **Tbsi_Get_TCG_Log** function returns the TCG Event Log for the system, and the buffer size depends on the number of events.

Windows 10:

The function may return a log that uses a format that is compatible with different hashing algorithms, depending on hardware capabilities and firmware settings. This log formats each event except the first as a `TCG_PCR_EVENT2` structure:

```
typedef struct {
    TCG_PCRINDEX PCRIndex;
    TCG_EVENTTYPE EventType;
    TPML_DIGEST_VALUES Digests;
    UINT32 EventSize;
    UINT8 Event[EventSize];
} TCG_PCR_EVENT2;

typedef struct {
    UINT32 Count;
    TPMT_HA Digests;
} TPML_DIGEST_VALUES;

typedef struct {
    UINT16 HashAlg;
    UINT8 Digest[size_varies_with_algorithm];
} TPMT_HA;
```

The log formats the first event as a `TCG_PCR_EVENT` structure, which is described later in this Remarks section. The following table describes the values of the members of this structure for this first event.

 Expand table

TCG_PCR_EVENT member	Value or description
PCRIndex	0
EventType	EV_NO_ACTION
Digest	20 bytes of zeros
EventSize	The size of the Event member
Event	Has a type of <code>TCG_EfiSpecIdEventStruct</code>

The following shows the syntax of the `TCG_EfiSpecIdEventStruct` structure that the **Event** member of the `TCG_PCR_EVENT` structure uses for the first log event.

```

typedef struct {
    BYTE[16] Signature;
    UINT32 PlatformClass;
    UINT8 SpecVersionMinor;
    UINT8 SpecVersionMajor;
    UINT8 SpecErrata;
    UINT8 UintNSize;
    UINT32 NumberOfAlgorithms;
    TCG_EfiSpecIdEventAlgorithmSize DigestSizes[NumberOfAlgorithms];
    UINT8 VendorInfoSize;
    UINT8 VendorInfo[VendorInfoSize];
} TCG_EfiSpecIdEventStruct;

typedef struct {
    UINT16 HashAlg;
    UINT16 DigestSize;
} TCG_EfiSpecIdEventAlgorithmSize;

```

The **Signature** member of the **TCG_EfiSpecIdEventStruct** structure is set to a null-terminated ASCII string of "Spec ID Event03" when the log uses the format that is compatible with different hashing algorithms. The **DigestSizes** array in this first event contains the digest sizes for the different hashing algorithms that the log uses. When a parser inspects an event of type **TCG_PCR_EVENT2**, the parser can parse the **TPML_DIGEST_VALUES** member without information about all of the hashing algorithms present. The digest sizes in the first event allow the parser to skip the correct number of bytes for the digests that are present.

If the **Signature** member is not set to a null-terminated ASCII string of "Spec ID Event03", then the events in the log are of type **TCG_PCR_EVENT**, and the **TCG_EfiSpecIdEventStruct** structure does not contain the **NumberOfAlgorithms** and **DigestSizes** members.

The log format that is compatible with different hashing algorithms allows the platform and operating system to use SHA1, SHA256, or other hashing algorithms. If the platform supports the SHA256 hashing algorithm and the uses the log format that is compatible with different hashing algorithms, the platform uses the SHA256 algorithm instead of SHA1.

Windows Vista with SP1 and Windows Server 2008: The function returns the log directly from the ACPI table and returns the entire ACPI allocated buffer, including the unused buffer after any events.

The Windows-defined events in the TCG event log are a tuple of {Type, Length, Value}. You can parse the log using the following **TCG_PCR_EVENT** structure from the [TCG PC](#)

[Client spec](#). You can create a correlation between lists of log events using the information in the [TPM PCP Toolkit](#) and the [TPM Main Specification](#).

```
typedef struct {
    TCG_PCRINDEX PCRIndex;
    TCG_EVENTTYPE EventType;
    TCG_DIGEST Digest;
    UINT32 EventSize;
    UINT8 Event[EventSize];
} TCG_PCR_EVENT;
```

The memory size required for the *pOutputBuf* parameter should either be the constant in **TBS_IN_OUT_BUF_SIZE_MAX**, defined in the Tbs.h header file, or it should be obtained by calling the **Tbsi_Get_TCG_Log** function with a zero length buffer to get the required buffer size.

Windows Vista with SP1 and Windows Server 2008: Calling the **Tbsi_Get_TCG_Log** function with a zero length buffer to get the required buffer size is not supported. We recommend that you use the constant **TBS_IN_OUT_BUF_SIZE_MAX**, defined in the Tbs.h header file, for the memory size for the *pOutputBuf* parameter.


Examples

C++

```
#include <windows.h>
#include <tbs.h>
#pragma comment(lib, "Tbs.lib")



void main()
{
    TBS_RESULT result;
    TBS_HCONTEXT hContext;
    TBS_CONTEXT_PARAMS contextParams;
    contextParams.version = TBS_CONTEXT_VERSION_ONE;
    result = Tbsi_Context_Create(&contextParams, &hContext);
    if (result == TBS_SUCCESS)
    {
        UINT32 iLogSize = TBS_IN_OUT_BUF_SIZE_MAX;
        BYTE* pLogBuffer = new BYTE[iLogSize];
        result = Tbsi_Get_TCG_Log(hContext, pLogBuffer, &iLogSize);
    }
}
```

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista with SP1 [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	tbs.h
Library	Tbs.lib
DLL	Tbs.dll

Feedback

Was this page helpful?  Yes  No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

Tbsi_Get_TCG_Log_Ex function (tbs.h)

Article03/15/2023

Gets the Windows Boot Configuration Log (WBCL), also referred to as the TCG log, of the specified type.

Syntax

C++

```
TBS_RESULT Tbsi_Get_TCG_Log_Ex(  
    [in]          UINT32  logType,  
    [out, optional] PBYTE  pbOutput,  
    [in, out]     PUINT32 pcbOutput  
);
```

Parameters

[in] logType

The type of log to retrieve.

 Expand table

Value	Meaning
TBS_TCGLOG_SRTM_CURRENT 0	The log associated with PCRs 0-15 for the current session (boot or resume).
TBS_TCGLOG_DRTM_CURRENT 1	The log associated with PCRs 17-22 for the current session (boot or resume).
TBS_TCGLOG_SRTM_BOOT 2	The log associated with PCRs 0-15 for the most recent clean boot session.
TBS_TCGLOG_SRTM_RESUME 3	The log associated with PCRs 0-15 for the most recent resume from hibernation.

[out, optional] pbOutput


Pointer to a buffer that receives and stores the WBCL. Set to **NULL** to estimate the required buffer when the location pointed to by *pcbOutput* is also 0 on input.

[in, out] pcbOutput

Pointer to an unsigned long integer that specifies the size, in bytes, of the output buffer. On success, contains the size, in bytes, of the data pointed to by *pOutput*. On failure, does not contain a value.

Note If *pbOutput* is **NULL** and the location pointed to by *pcbOutput* is 0, the function returns **TBS_E_BUFFER_TOO_SMALL**. In that case, *pcbOutput* will point to the required size of *pbOutput*.

Return value

 Expand table

Return code/value	Description
TBS_SUCCESS 0 (0x0)	The function succeeded.
TBS_E_NO_EVENT_LOG 1 (0x1)	TBS_TCGLOG_DRTM_CURRENT was requested but DRTM was not enabled on the system when the system booted.
TBS_E_INTERNAL_ERROR 2150121473 (0x80284001)	An internal software error occurred. <div>Note If TBS_E_INTERNAL_ERROR is returned, the system event log may contain event ID 16385 from the TBS event source with error code 0x80070032. This may indicate that the hardware platform does not provide a TCG event log to the operating system. Sometimes this can be resolved by installing a BIOS upgrade from the platform manufacturer.</div>
TBS_E_INVALID_OUTPUT_POINTER 2150121475 (0x80284003)	A specified output pointer is not valid.
TBS_E_INSUFFICIENT_BUFFER 2150121477 (0x80284005)	The output buffer is too small.
TBS_E_BUFFER_TOO_LARGE 2150121486 (0x8028400E)	The output buffer is too large.
TBS_E_TPM_NOT_FOUND 2150121487 (0x8028400F)	A compatible Trusted Platform Module (TPM) Security Device cannot be found on this computer.

TBS_E_DEACTIVATED
2150121494 (0x80284016)

The Trusted Platform Module (TPM) Security Device is deactivated.

Remarks

The `Tbsi_Get_TCG_Log_Ext` function returns the TCG Event Log for the system, and the buffer size depends on the number of events.


The function may return a log that uses a format that is compatible with different hashing algorithms, depending on hardware capabilities and firmware settings. This log formats each event except the first as a `TCG_PCR_EVENT2` structure:

```
typedef struct {
    TCG_PCRINDEX PCRIndex;
    TCG_EVENTTYPE EventType;
    TPML_DIGEST_VALUES Digests;
    UINT32 EventSize;
    UINT8 Event[EventSize];
} TCG_PCR_EVENT2;

typedef struct {
    UINT32 Count;
    TPMT_HA Digests;
} TPML_DIGEST_VALUES;

typedef struct {
    UINT16 HashAlg;
    UINT8 Digest[size_varies_with_algorithm];
} TPMT_HA;
```

The log formats the first event as a `TCG_PCR_EVENT` structure, which is described later in this Remarks section. The following table describes the values of the members of this structure for this first event.

 Expand table

TCG_PCR_EVENT member	Value or description
PCRIndex	0
EventType	EV_NO_ACTION
Digest	20 bytes of zeros

EventSize	The size of the Event member
Event	Has a type of TCG_EfiSpecIdEventStruct

The following shows the syntax of the **TCG_EfiSpecIdEventStruct** structure that the **Event** member of the **TCG_PCR_EVENT** structure uses for the first log event.

```
typedef struct {
    BYTE[16] Signature;
    UINT32 PlatformClass;
    UINT8 SpecVersionMinor;
    UINT8 SpecVersionMajor;
    UINT8 SpecErrata;
    UINT8 UintNSize;
    UINT32 NumberOfAlgorithms;
    TCG_EfiSpecIdEventAlgorithmSize DigestSizes[NumberOfAlgorithms];
    UINT8 VendorInfoSize;
    UINT8 VendorInfo[VendorInfoSize];
} TCG_EfiSpecIdEventStruct;

typedef struct {
    UINT16 HashAlg;
    UINT16 DigestSize;
} TCG_EfiSpecIdEventAlgorithmSize;
```

The **Signature** member of the **TCG_EfiSpecIdEventStruct** structure is set to a null-terminated ASCII string of "Spec ID Event03" when the log uses the format that is compatible with different hashing algorithms. The **DigestSizes** array in this first event contains the digest sizes for the different hashing algorithms that the log uses. When a parser inspects an event of type **TCG_PCR_EVENT2**, the parser can parse the **TPML_DIGEST_VALUES** member without information about all of the hashing algorithms present. The digest sizes in the first event allow the parser to skip the correct number of bytes for the digests that are present.

If the **Signature** member is not set to a null-terminated ASCII string of "Spec ID Event03", then the events in the log are of type **TCG_PCR_EVENT**, and the **TCG_EfiSpecIdEventStruct** structure does not contain the **NumberOfAlgorithms** and **DigestSizes** members.

The log format that is compatible with different hashing algorithms allows the platform and operating system to use SHA1, SHA256, or other hashing algorithms. If the platform supports the SHA256 hashing algorithm and the uses the log format that is compatible

with different hashing algorithms, the platform uses the SHA256 algorithm instead of SHA1.

The Windows-defined events in the TCG event log are a tuple of {Type, Length, Value}. You can parse the log using the following TCG_PCR_EVENT structure from the [TCG PC Client spec](#). You can create a correlation between lists of log events using the information in the [TPM PCP Toolkit](#) and the [TPM Main Specification](#).

```
typedef struct {
    TCG_PCRINDEX PCRIndex;
    TCG_EVENTTYPE EventType;
    TCG_DIGEST Digest;
    UINT32 EventSize;
    UINT8 Event[EventSize];
} TCG_PCR_EVENT;
```

The memory size required for the *pOutputBuf* parameter should either be the constant in **TBS_IN_OUT_BUF_SIZE_MAX**, defined in the Tbs.h header file, or it should be obtained by calling the **Tbsi_Get_TCG_Log_Ex** function with a zero length buffer to get the required buffer size.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1803 [desktop apps only]
Minimum supported server	Windows Server [desktop apps only]
Target Platform	Windows
Header	tbs.h
Library	Tbs.lib
DLL	Tbs.dll

Feedback

Was this page helpful?



Yes



No

[Provide product feedback](#) 

| [Get help at Microsoft Q&A](#)

Tbsi_Physical_Presence_Command function (tbs.h)

Article 02/22/2024

Passes a physical presence ACPI command through TBS to the driver.

Syntax

C++

```
TBS_RESULT Tbsi_Physical_Presence_Command(  
    [in]      TBS_HCONTEXT hContext,  
    [in]      PCBYTE       pabInput,  
    [in]      UINT32       cbInput,  
    [out]     PBYTE        pabOutput,  
    [in, out] PUINT32      pcbOutput  
);
```

Parameters

[in] hContext

The context of the ACPI command.

[in] pabInput

A pointer to a buffer that contains the input to the ACPI command.

The input to the ACPI command is defined in the *TCG Physical Presence Interface Specification* at <https://www.trustedcomputinggroup.org>. The buffer should contain *Arg2* and *Arg3* values as defined in this document. The values for *Arg0* and *Arg1* are static and automatically added. For example, if this method is used for Get Physical Presence Interface Version, then *Arg2* is the integer value 1 and *Arg3* is empty, so the buffer should just contain an integer value of 1. If this method is used for "Submit TPM Operation Request to Pre-OS Environment", then *Arg2* is the integer value 2 and *Arg3* will be the integer for the specified operation, such as 1 for enable or 2 for disable.

[in] cbInput

The length, in bytes, of the input buffer.

[out] pabOutput

A pointer to a buffer to contain the output of the ACPI command.

The buffer will contain the return value from the command as defined in the [TCG Physical Presence Interface Specification](#) .

[in, out] pcbOutput

A pointer to an unsigned long integer 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*. If the function fails, this parameter does not receive a value.

Return value

If the function succeeds, the function returns TBS_SUCCESS.

If the function fails, it returns a TBS return code that indicates the error.


 Expand table

Return code/value	Description
TBS_SUCCESS 0 (0x0)	The function was successful.
TBS_E_BAD_PARAMETER 2150121474 (0x80284002)	One or more parameter values are not valid.
TBS_E_INTERNAL_ERROR 2150121473 (0x80284001)	An internal software error occurred.
TBS_E_INVALID_CONTEXT_PARAM 2150121479 (0x80284007)	A context parameter that is not valid was passed when attempting to create a TBS context.
TBS_E_INVALID_OUTPUT_POINTER 2150121475 (0x80284003)	A specified output pointer is not valid.

Remarks

For more information, see [TCG Physical Presence Interface Specification](#) .

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	tbs.h
Library	Tbs.lib
DLL	Tbs.dll

Feedback

Was this page helpful?



[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

Tbsi_Revoke_Attestation function (tbs.h)

Article 10/05/2021


Invalidates the PCRs if the ELAM driver detects a policy-violation (a rootkit, for example).

Syntax

C++

```
TBS_RESULT Tbsi_Revoke_Attestation();
```

Return value

 Expand table

Return code/value	Description
TBS_SUCCESS 0 (0x0)	The function succeeded.
TBS_E_INTERNAL_ERROR 2150121473 (0x80284001)	An internal software error occurred. <div>Note If TBS_E_INTERNAL_ERROR is returned, the system event log may contain event ID 16385 from the TBS event source with error code 0x80070032. This may indicate that the hardware platform does not provide a TCG event log to the operating system. Sometimes this can be resolved by installing a BIOS upgrade from the platform manufacturer.</div>

Remarks

This function is callable from kernel mode.

You must run this function with administrative rights. This function extends PCR[12] by an unspecified value and increment 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. Since 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 computer goes to hibernation and subsequently resumes, the previous PCR extent will be lost, and the broken trust will not be reflected in the PCR measurements anymore. To address this, the **Tbsi_Revoke_Attestation** function 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 you can't construct the missing WBCL after the fact.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	tbs.h
Library	Tbs.lib
DLL	Tbs.dll

Feedback

Was this page helpful?

☐ Yes

☐ No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Tbsip_Cancel_Commands function (tbs.h)

Article02/22/2024

Cancels all outstanding commands for the specified context.

Syntax

C++

```
TBS_RESULT Tbsip_Cancel_Commands(  
    [in] TBS_HCONTEXT hContext  
);
```

Parameters


[in] hContext

A TBS handle to the context whose commands are to be canceled and that was obtained from previous call to the [Tbsi_Context_Create](#) function.

Return value

If the function succeeds, the function returns TBS_SUCCESS.

If the function fails, it returns a TBS return code that indicates the error.

 Expand table

Return code/value	Description
TBS_SUCCESS 0 (0x0)	The function was successful.
TBS_E_INTERNAL_ERROR 2150121473 (0x80284001)	An internal software error occurred.
TBS_E_INVALID_CONTEXT 2150121476 (0x80284004)	The specified context handle does not refer to a valid context.
TBS_E_IOERROR	An error occurred while communicating with the TPM.

Remarks

When a command is canceled, TBS sends a message to the command that indicates that the command was canceled.

Requirements

[Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	tbs.h
Library	Tbs.lib
DLL	Tbs.dll

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

Tbsip_Context_Close function (tbs.h)

Article 10/13/2021

Closes a context handle, which releases resources associated with the context in TBS and closes the binding handle used to communicate with TBS.

Syntax

C++

```
TBS_RESULT Tbsip_Context_Close(  
    [in] TBS_HCONTEXT hContext  
);
```

Parameters

[in] hContext

A handle of the context to be closed.

Return value

If the function succeeds, the function returns TBS_SUCCESS.

If the function fails, it returns a TBS return code that indicates the error.


 Expand table

Return code/value	Description
TBS_SUCCESS 0 (0x0)	The function was successful.
TBS_E_INTERNAL_ERROR 2150121473 (0x80284001)	An internal software error occurred.
TBS_E_INVALID_CONTEXT 2150121476 (0x80284004)	The specified context handle does not refer to a valid context.

Remarks

When the context handle is closed, the structure associated with the context handle is zeroed, which ensures that subsequent attempts to use the handle will result in an error. All objects that have been created under this context will be flushed.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	tbs.h
Library	Tbs.lib
DLL	Tbs.dll

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

Tbsip_Submit_Command function (tbs.h)

Article 10/13/2021

Submits a Trusted Platform Module (TPM) command to TPM Base Services (TBS) for processing.

Syntax

C++

```
TBS_RESULT Tbsip_Submit_Command(  
    [in]      TBS_HCONTEXT      hContext,  
    [in]      TBS_COMMAND_LOCALITY Locality,  
    [in]      TBS_COMMAND_PRIORITY Priority,  
    [in]      PCBYTE            pabCommand,  
    [in]      UINT32             cbCommand,  
    [out]     PBYTE             pabResult,  
    [in, out] PUINT32           pcbResult  
);
```

Parameters

[in] hContext

The handle of the context that is submitting the command.

[in] Locality

Used to set the locality for the TPM command. This must be one of the following values.


[Expand table](#)

Value	Meaning
TBS_COMMAND_LOCALITY_ZERO 0 (0x0)	Locality zero. This is the only locality currently supported.
TBS_COMMAND_LOCALITY_ONE 1 (0x1)	Locality one.
TBS_COMMAND_LOCALITY_TWO 2 (0x2)	Locality two.

TBS_COMMAND_LOCALITY_THREE 3 (0x3)	Locality three.
TBS_COMMAND_LOCALITY_FOUR 4 (0x4)	Locality four.

[in] **Priority**

The priority level that the command should have. This parameter can be one of the following values.

 **Expand table**

Value	Meaning
TBS_COMMAND_PRIORITY_LOW 100 (0x64)	Used for low priority application use.
TBS_COMMAND_PRIORITY_NORMAL 200 (0xC8)	Used for normal priority application use.
TBS_COMMAND_PRIORITY_SYSTEM 400 (0x190)	Used for system tasks that access the TPM.
TBS_COMMAND_PRIORITY_HIGH 300 (0x12C)	Used for high priority application use.
TBS_COMMAND_PRIORITY_MAX 2147483648 (0x80000000)	Used for tasks that originate from the power management system.

[in] **pabCommand**

A pointer to a buffer that contains the TPM command to process.

[in] **cbCommand**

The length, in bytes, of the command.

[out] **pabResult**

A pointer to a buffer to receive the result of the TPM command. This buffer can be the same as *pabCommand*.

[in, out] **pcbResult**


An integer that, on input, specifies the size, in bytes, of the result buffer. This value is set when the submit command returns. If the supplied buffer is too small, this parameter, on output, is set to the required size, in bytes, for the result.

Return value

If the function succeeds, the function returns TBS_SUCCESS.

A command can be submitted successfully and still fail at the TPM. In this case, the failure code is returned as a standard TPM error in the result buffer.

If the function fails, it returns a TBS return code that indicates the error.

 Expand table

Return code/value	Description
TBS_SUCCESS 0 (0x0)	The function was successful.
TBS_E_BAD_PARAMETER 2150121474 (0x80284002)	One or more parameter values are not valid.
TBS_E_BUFFER_TOO_LARGE 2150121486 (0x8028400E)	The input or output buffer is too large.
TBS_E_INTERNAL_ERROR 2150121473 (0x80284001)	An internal software error occurred.
TBS_E_INSUFFICIENT_BUFFER 2150121477 (0x80284005)	The specified output buffer is too small.
TBS_E_INVALID_CONTEXT 2150121476 (0x80284004)	The specified context handle does not refer to a valid context.
TBS_E_INVALID_OUTPUT_POINTER 2150121475 (0x80284003)	A specified output pointer is not valid.
TBS_E_IOERROR 2150121478 (0x80284006)	An error occurred while communicating with the TPM.

Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	tbs.h
Library	Tbs.lib
DLL	Tbs.dll

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)  | [Get help at Microsoft Q&A](#)

TBS Return Codes

Article • 12/11/2020

TBS uses the following codes to indicate the result of a function call.

Constant/value	Description
TBS_SUCCESS 0 (0x0)	The function succeeded.
TBS_E_INTERNAL_ERROR 2150121473 (0x80284001)	An internal software error occurred.
TBS_E_BAD_PARAMETER 2150121474 (0x80284002)	One or more parameter values are not valid.
TBS_E_INVALID_OUTPUT_POINTER 2150121475 (0x80284003)	A specified output pointer is bad.
TBS_E_INVALID_CONTEXT 2150121476 (0x80284004)	The specified context handle does not refer to a valid context.
TBS_E_INSUFFICIENT_BUFFER 2150121477 (0x80284005)	The specified output buffer is too small.
TBS_E_IOERROR 2150121478 (0x80284006)	An error occurred while communicating with the TPM.
TBS_E_INVALID_CONTEXT_PARAM 2150121479 (0x80284007)	A context parameter that is not valid was passed when attempting to create a TBS context.
TBS_E_SERVICE_NOT_RUNNING 2150121480 (0x80284008)	The TBS service is not running and could not be started.
TBS_E_TOO_MANY_TBS_CONTEXTS 2150121481 (0x80284009)	A new context could not be created because there are too many open contexts.
TBS_E_TOO_MANY_RESOURCES 2150121482 (0x8028400A)	A new virtual resource could not be created because there are too many open virtual resources.
TBS_E_SERVICE_START_PENDING 2150121483 (0x8028400B)	The TBS service has been started but is not yet running.
TBS_E_PPI_NOT_SUPPORTED 2150121484 (0x8028400C)	The physical presence interface is not supported.
TBS_E_COMMAND_CANCELED 2150121485 (0x8028400D)	The command was canceled.

Constant/value	Description
TBS_E_BUFFER_TOO_LARGE 2150121486 (0x8028400E)	The input or output buffer is too large.
TBS_E_TPM_NOT_FOUND 2150121487 (0x8028400F)	A compatible Trusted Platform Module (TPM) Security Device cannot be found on this computer.
TBS_E_SERVICE_DISABLED 2150121488 (0x80284010)	The TBS service has been disabled.
TBS_E_NO_EVENT_LOG 2150121489 (0x80284011)	The TBS event log is not available.
TBS_E_ACCESS_DENIED 2150121490 (0x80284012)	The caller does not have the appropriate rights to perform the requested operation.
TBS_E_PROVISIONING_NOT_ALLOWED 2150121491 (0x80284013)	The TPM provisioning action is not allowed by the specified flags.
TBS_E_PPI_FUNCTION_UNSUPPORTED 2150121492 (0x80284014)	The Physical Presence Interface of this firmware does not support the requested method.
TBS_E_OWNERAUTH_NOT_FOUND 2150121493 (0x80284015)	The requested TPM OwnerAuth value was not found.
TBS_E_PROVISIONING_INCOMPLETE 2150121493 (0x80284015)	The TPM provisioning did not complete. For more information on completing the provisioning, call the Win32_Tpm WMI method for provisioning the TPM ('Provision') and check the returned information.

Requirements

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	Tbs.h

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

TBS Structures

Article • 02/26/2020

TBS supports the following structures.

In this section

Topic	Description
TBS_CONTEXT_PARAMS	Specifies the version of the TBS context implementation.
TBS_CONTEXT_PARAMS2	Specifies the version of the TBS context implementation. You must use this structure if your application works with both versions of TPM.
TPM_DEVICE_INFO	Provides information about the version of the TPM.

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

TBS_CONTEXT_PARAMS structure (tbs.h)

Article 10/05/2021

Specifies the version of the TBS context implementation.

To connect to TBS, the client must run as administrator. TBS also limits access to locality ZERO.

Syntax

C++

```
typedef struct tdTBS_CONTEXT_PARAMS {  
    UINT32 version;  
} TBS_CONTEXT_PARAMS, *PTBS_CONTEXT_PARAMS;
```

Members

version

The version of the TBS context implementation. This parameter must be TBS_CONTEXT_VERSION_ONE.

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	tbs.h

Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

TBS_CONTEXT_PARAMS2 structure (tbs.h)

Article 01/27/2022

Specifies the version of the TBS context implementation. You must use this structure if your application works with both versions of TPM.

Applications interacting with just TPM 2.0 should pass a pointer to a **TBS_CONTEXT_PARAMS2** structure, with **version** set to **TPM_VERSION_20**, and **includeTpm20** set to 1.

Applications interacting with both TPM 1.2 and TPM 2.0 should pass a pointer to a **TBS_CONTEXT_PARAMS2** structure, with **version** set to **TPM_VERSION_20**, **includeTpm20** set to 1, and **includeTpm12** set to 1.

Syntax

C++

```
typedef struct tdTBS_CONTEXT_PARAMS2 {
    UINT32 version;
    union {
        struct {
            UINT32 requestRaw : 1;
            UINT32 includeTpm12 : 1;
            UINT32 includeTpm20 : 1;
        };
        UINT32 asUINT32;
    };
} TBS_CONTEXT_PARAMS2, *PTBS_CONTEXT_PARAMS2;
```

Members

version

The version of the TBS context implementation. This must be set to **TPM_VERSION_20**.

requestRaw

includeTpm12

includeTpm20

asUINT32

Used to access all of the bits in one variable.

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	tbs.h

Feedback

Was this page helpful?

[Get help at Microsoft Q&A](#)

TPM_DEVICE_INFO structure (tbs.h)

Article 10/05/2021

Provides information about the version of the TPM.

Syntax

C++

```
typedef struct _TPM_DEVICE_INFO {
    UINT32 structVersion;
    UINT32 tpmVersion;
    UINT32 tpmInterfaceType;
    UINT32 tpmImpRevision;
} TPM_DEVICE_INFO, *PTPM_DEVICE_INFO;
```

Members

structVersion

The version of the TBS context implementation. This parameter must be set to TPM_VERSION_20.

tpmVersion

TPM version. Will be set to TPM_VERSION_12 or TPM_VERSION_20.

tpmInterfaceType

Reserved

tpmImpRevision

Reserved

Requirements

Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]

Header	tbs.h

Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)